CESE4015 Software Systems

Unified Modeling Language: An Introduction (Part 2)

Guohao Lan

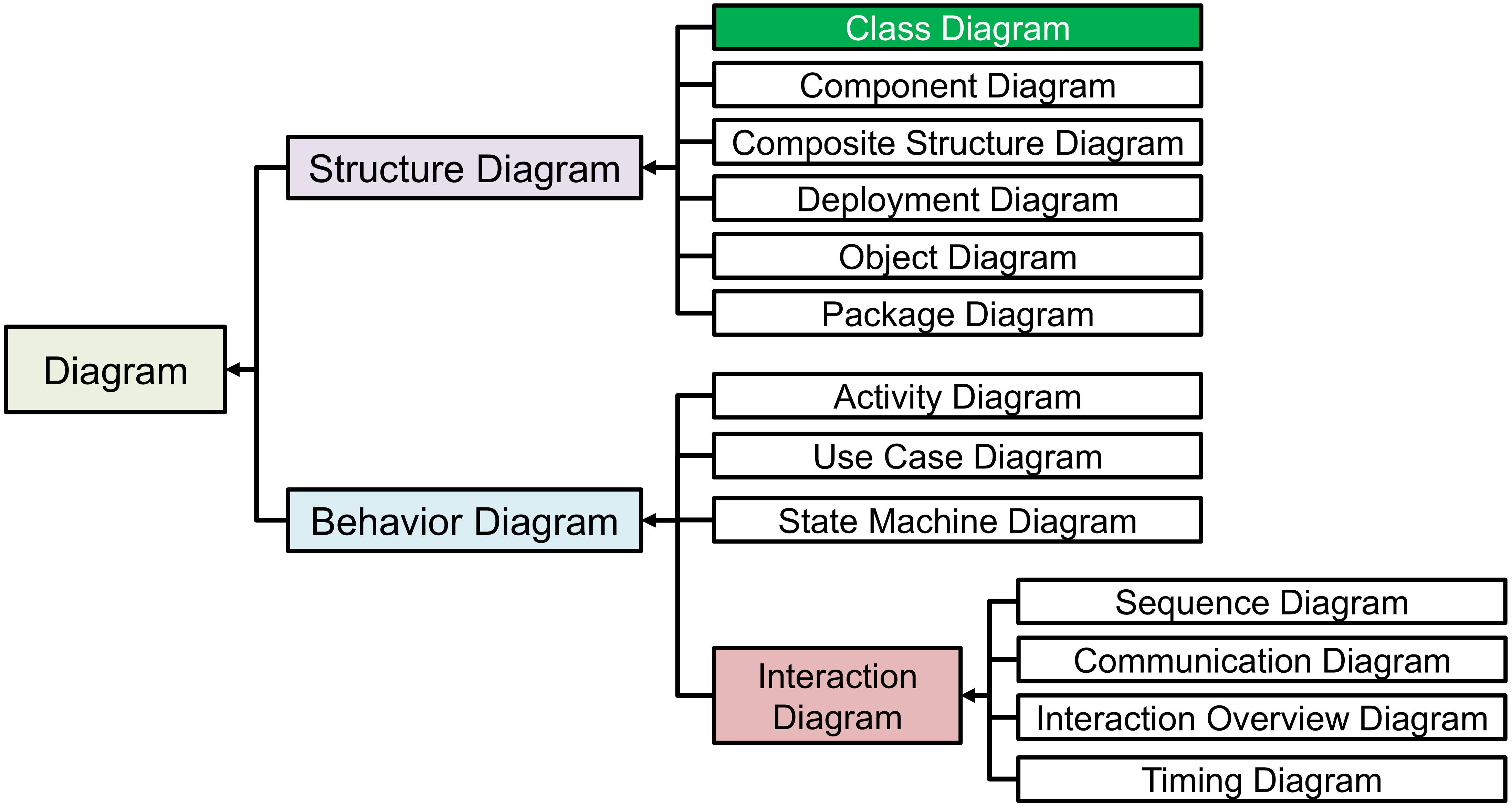
Embedded Systems Group

December 21th 2023

Agenda for UML

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

Class Diagram



Class Diagram

- What is a class diagram?

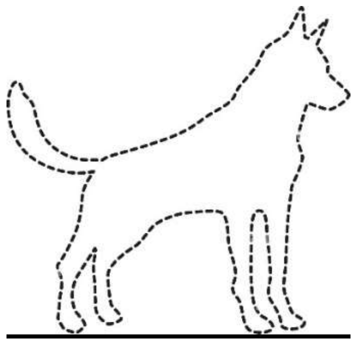
  ■ **Class Diagram:** describes the structure of classes in the system and the various kinds of static relationships among them.

Class Diagram (cont.)

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

Class

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



Create instance →

Object

Properties

Color: Flesh

Height: 15cm

Length: 30cm

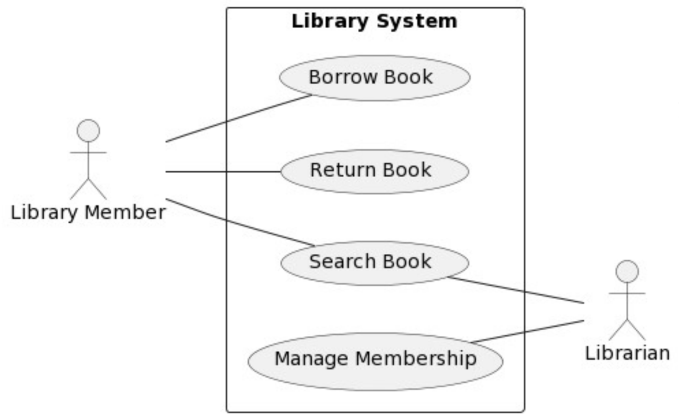
Weight: 2kg

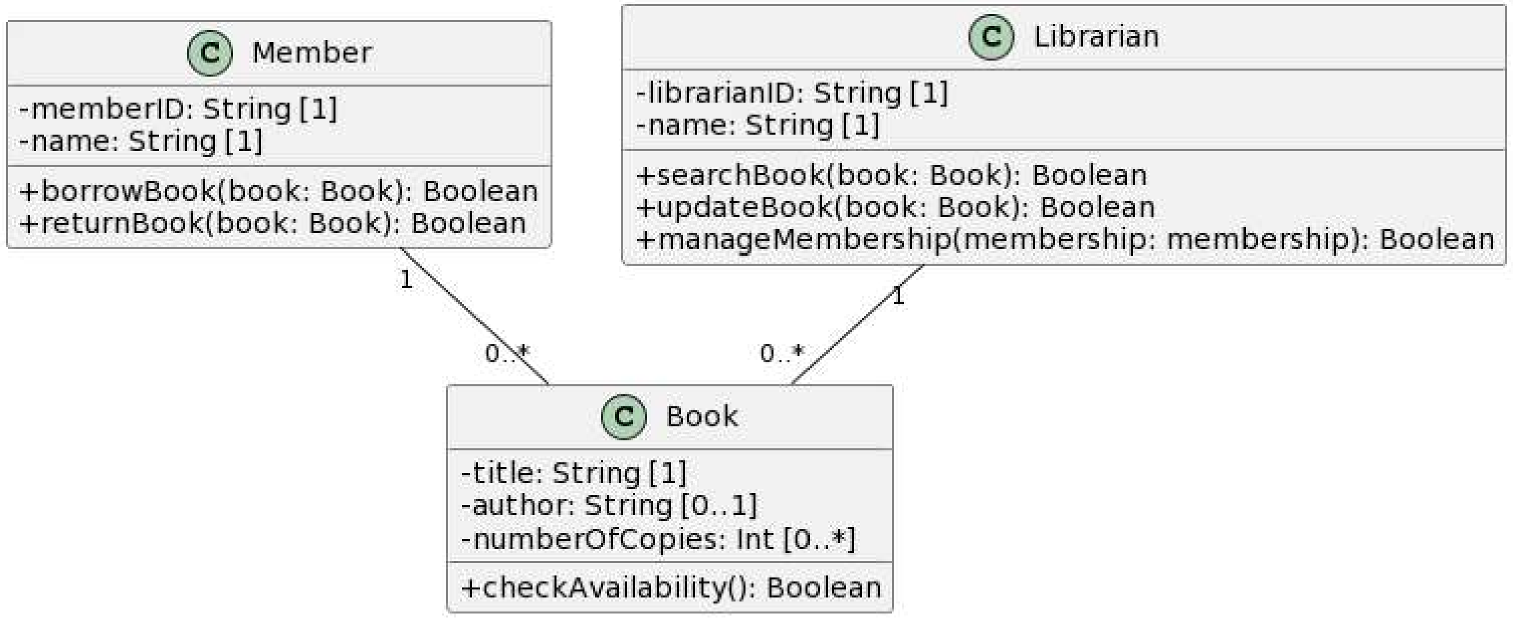
  - Object-Orientation “features” in Rust:
    - Using traits to define shared behavior in an abstract way.
    - Using struct to achieve the purpose of class:
    - References: *https://doc.rust-lang.org/book/ch17-02-trait-objects.html*
    - *https://jimmco.medium.com/classes-in-rust-c5b72c0f0a4c*

Class Diagram (cont.)

- **Discussion**:
  - What do you see in this diagram?
  - What are the elements in this diagram?
  - What message(s) this diagram may try to deliver?

Simplified Use Case




C Member
-memberID: String [1]
-name: String [1]
+borrowBook(book: Book): Boolean
+returnBook(book: Book): Boolean
C Librarian
-librarianID: String [1]
-name: String [1]
+searchBook(book: Book): Boolean
+updateBook(book: Book): Boolean
+manageMembership(membership: membership): Boolean
1
0..*
1
0..*
C Book
-title: String [1]
-author: String [0..1]
-numberOfCopies: Int [0..*]
+checkAvailability(): Boolean

Think → Pair → Share

Class Diagram (cont.)

- What is a class diagram?

■ **Class Diagram:** describes the structure of classes in the system and the various kinds of static relationships among them.

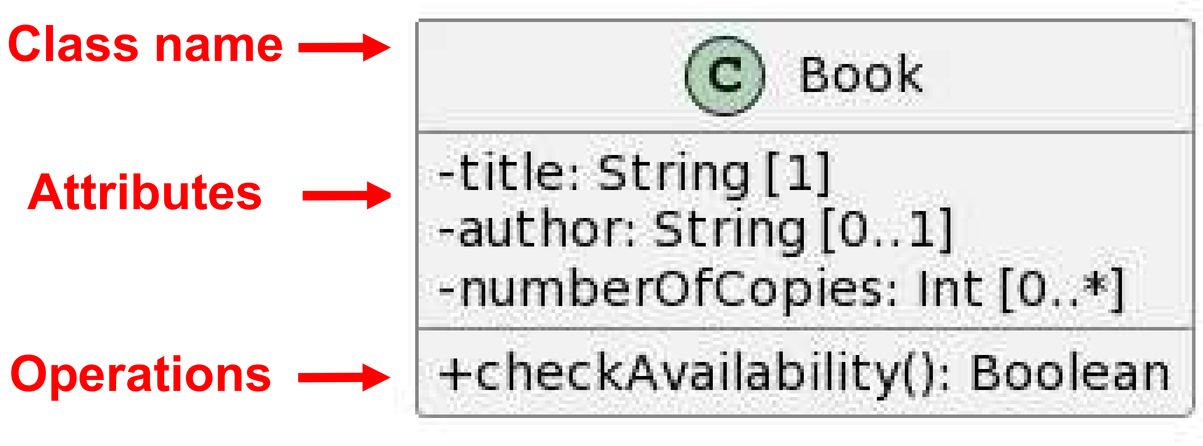
– It visualizes:
  - the static properties and operations of classes:
    – Attributes, methods, and associations.

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

Class Diagram (cont.)

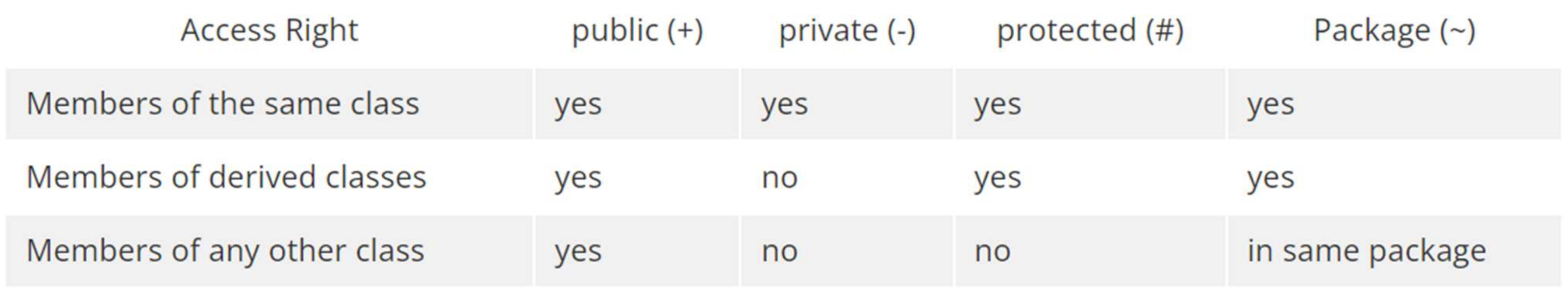
- Diagram of one class:
  - **Class notation:** contains three parts - class name, attributes, and operations.

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



Class 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

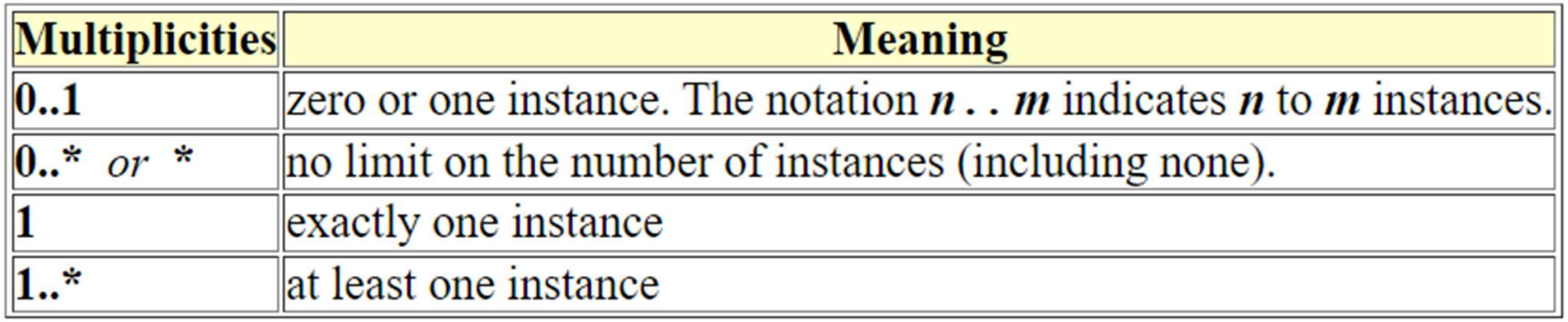


Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

Class Diagram (cont.)

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

– (2) Multiplicity:

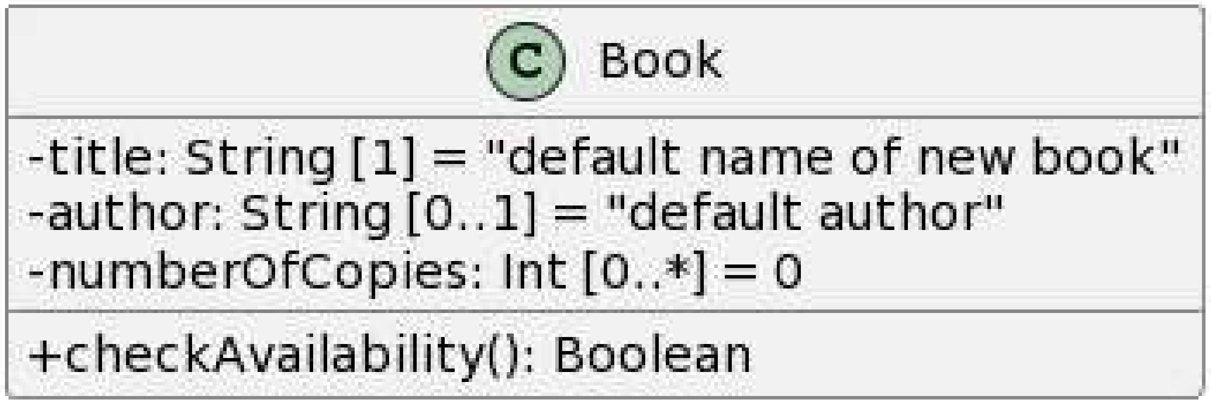


Multiplicities	**Meaning**
0..1	zero or one instance. The notation ***n* . . *m*** indicates ***n*** to ***m*** instances.
0..* *or* *****	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

Class Diagram (cont.)

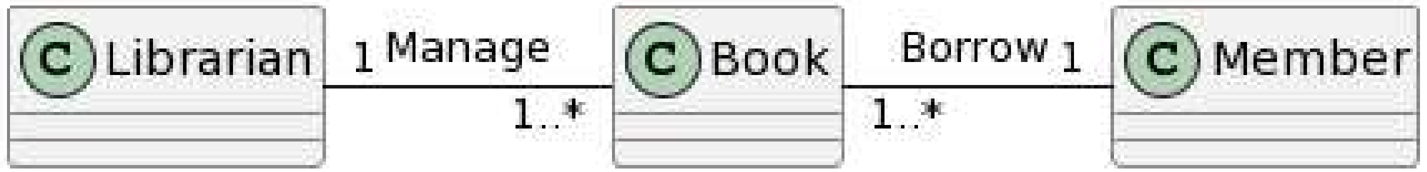
- Class attributes:
  - **Syntax:**
    visibility name : data_type [multiplicity] = default_value
- Class operations:
  - **Syntax:**
    visibility name (parameter-list) : return-type

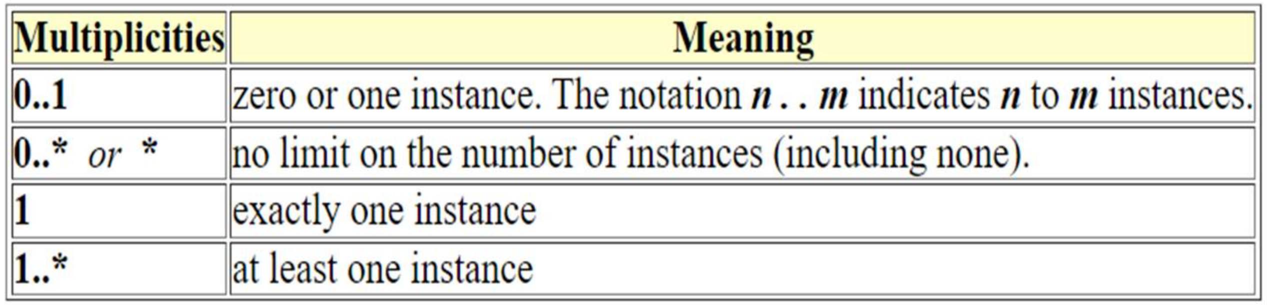
  – An example:


C
Book
-title: String [1] = "default name of new book"
-author: String [0..1] = "default author"
-numberOfCopies: Int [0..*] = 0
+checkAvailability(): Boolean

Class Diagram (cont.)

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



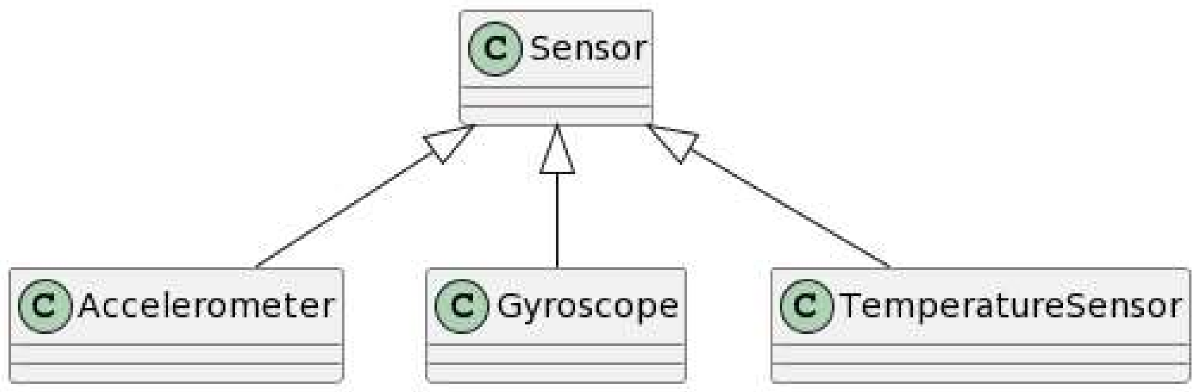


Multiplicities	Meaning
0..1	zero or one instance. The notation ***n . . m*** indicates ***n*** to ***m*** instances.
0..* *or* *****	no limit on the number of instances (including none).
1	exactly one instance
1..*	at least one instance

Class Relationship (cont.)

- Class relationships:

- **Discussion**:
  - (1) In the diagram below, you can see **solid lines with a hollow arrowhead** that points from one class to another class:

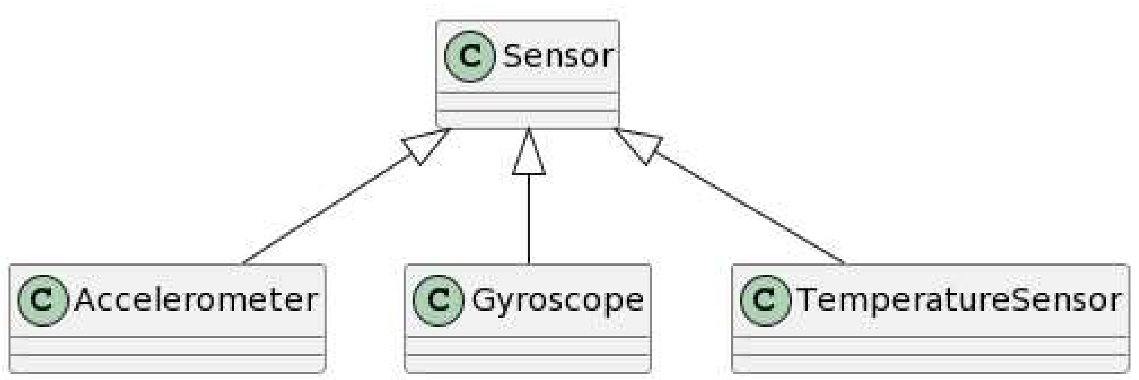


Attributes and operations of the classes are omitted

  - what relationship could this arrowed line indicate?
    - What is the relationship between **Sensor** and **Accelerometer**?

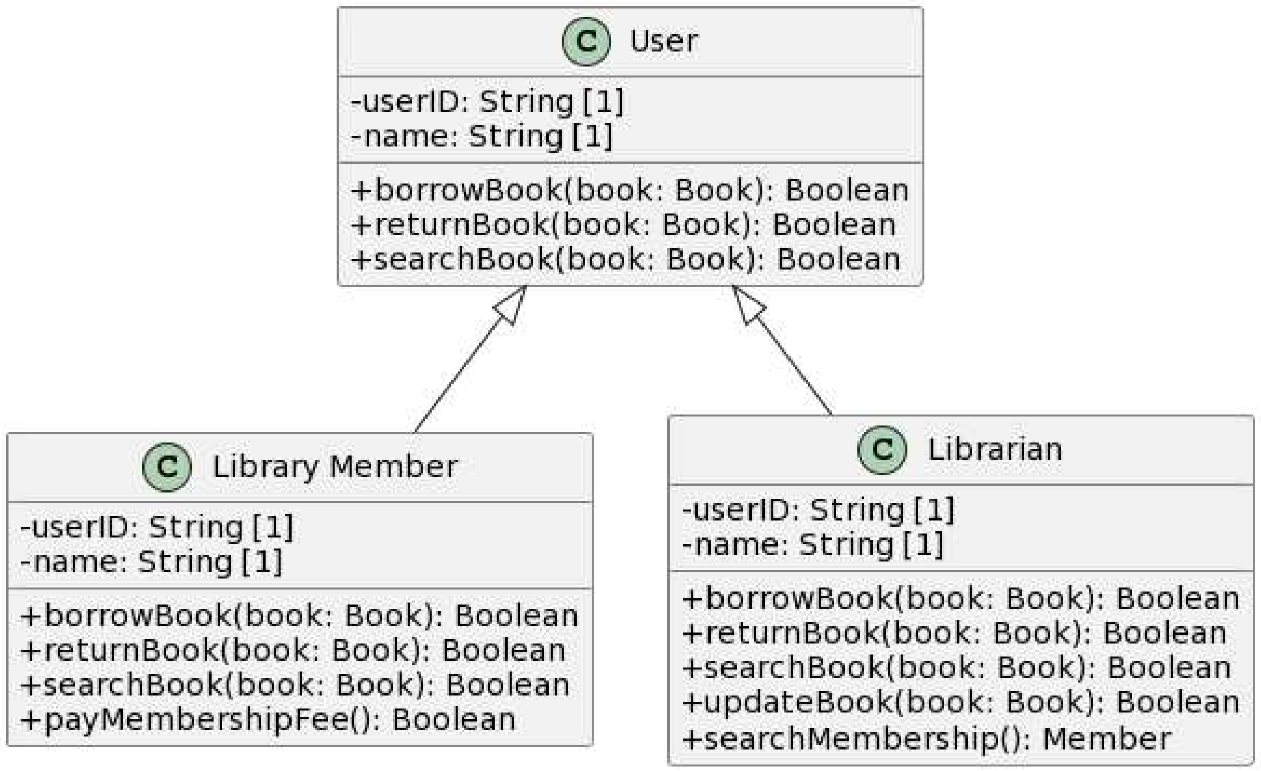
Class Relationship (cont.)

- 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



Class Relationship (cont.)

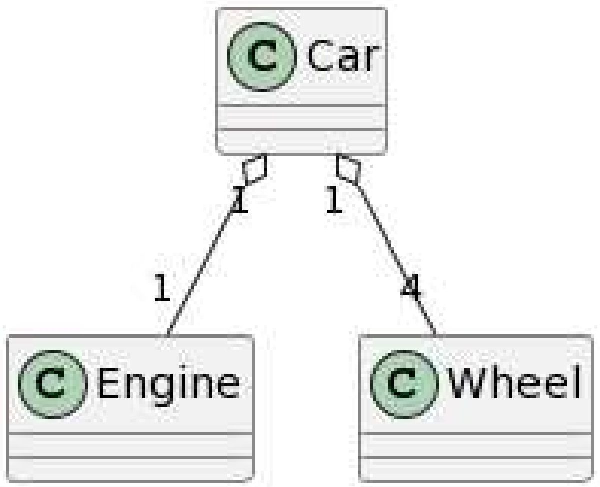
- Class relationships:
  - **Generalization:** an inheritance relationship
    - Represents an "is-a" relationship
    - A solid line with a hollow arrowhead that points from the child to the parent class



User
-userID: String [1]
-name: String [1]
+borrowBook(book: Book): Boolean
+returnBook(book: Book): Boolean
+searchBook(book: Book): Boolean
Library Member
-userID: String [1]
-name: String [1]
+borrowBook(book: Book): Boolean
+returnBook(book: Book): Boolean
+searchBook(book: Book): Boolean
+payMembershipFee(): Boolean
Librarian
-userID: String [1]
-name: String [1]
+borrowBook(book: Book): Boolean
+returnBook(book: Book): Boolean
+searchBook(book: Book): Boolean
+updateBook(book: Book): Boolean
+searchMembership(): Member

Class Relationship (cont.)

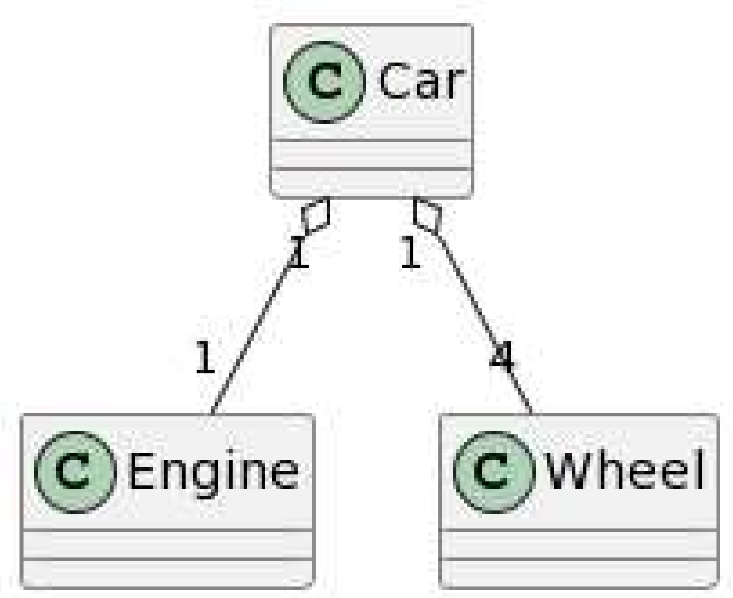
- **Discussion**:
  - (2) In the diagram below, you can see solid lines with an **unfilled diamond** that points from one class to the other classes:



  - What relationship could this type of line indicate?
    - What is the relationship between **Car** and **Engine**?
    - What is the relationship between **Car** and **Wheel**?

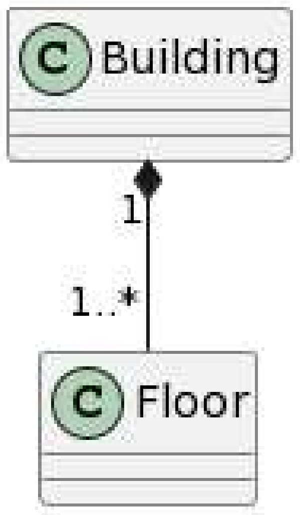
Class Relationship (cont.)

- 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 (independent).



Class Relationship (cont.)

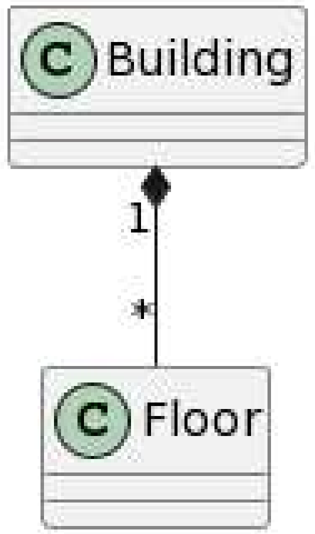
- **Discussion**:
  - (3) In the diagram below, you can see a solid line with a **filled diamond** that points from one class to the other:



  - What relationship could this type of line indicate?
    - What is the relationship between **Building** and **Floor**?
    - Why couldn't we use the aggregation relationship?

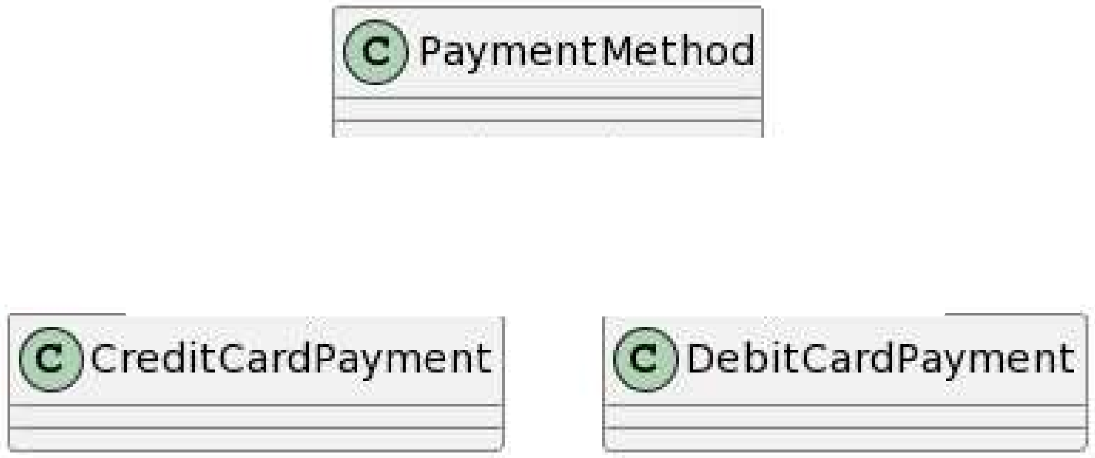
Class Relationship (cont.)

- 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.



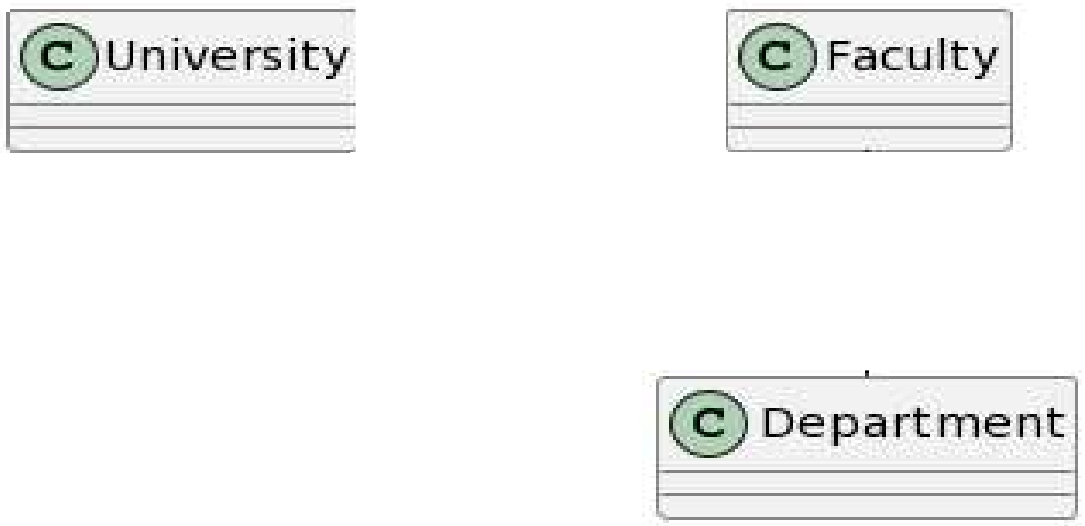
Class Relationship (cont.)

- Putting all together:
  - **Exercise #1**:
    - You are designing the payment module of a shopping system. You need design two payment methods, i.e., ***credit card*** and ***debit card*** payment, that may have some overleaps in features.
    - What do you think could be the relationships among the three classes below?



Class Relationship (cont.)

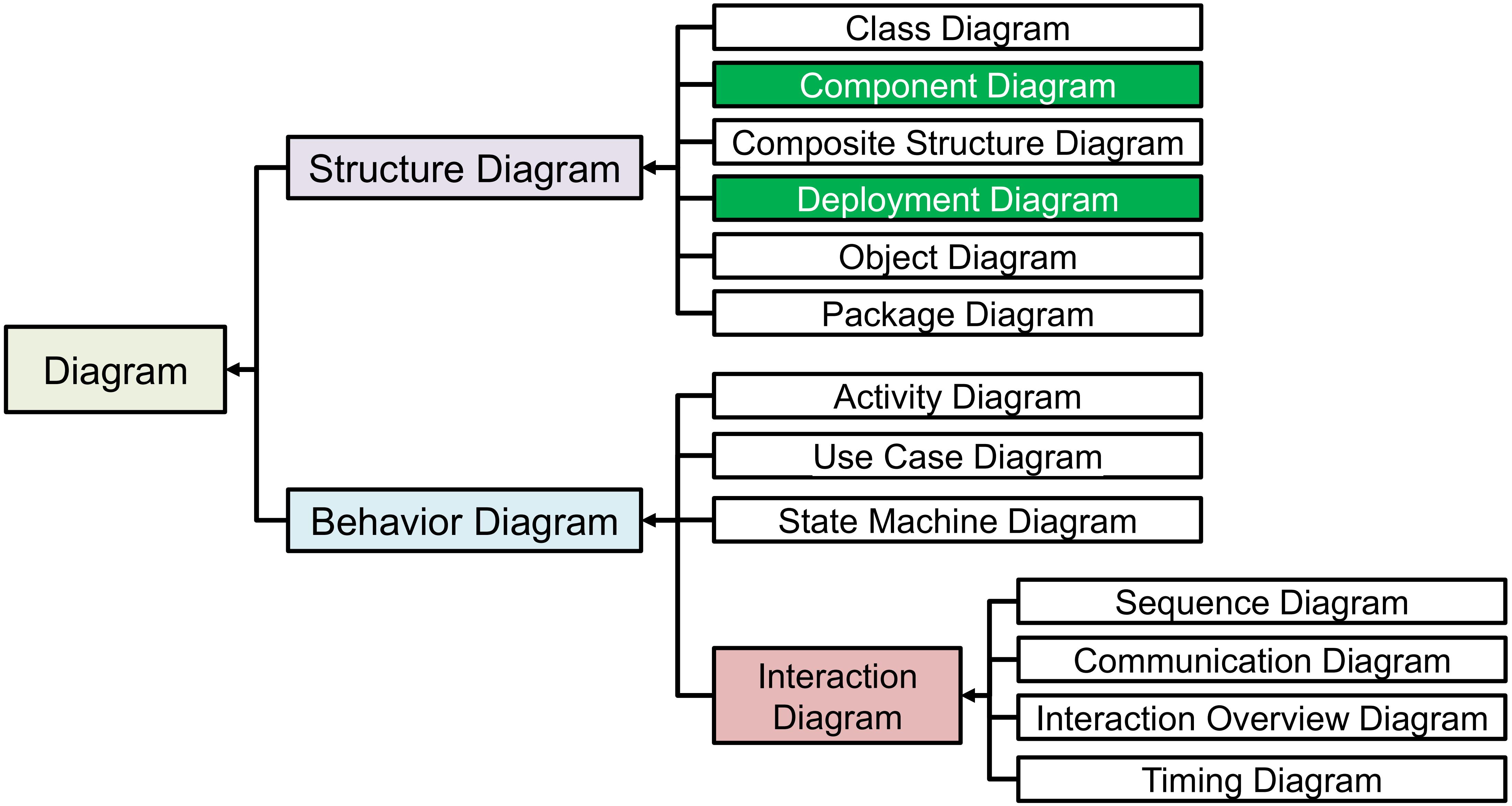
- Putting all together:
  - **Exercise #2**:
    - You are modeling the relationship between university, faculty, and departments. What do you think could be the relationships among the three classes below?



Class Diagram

- Short summary:
  - **Class Diagram:** describes the structure of classes in the system and the various kinds of static relationships 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.

Component and Deployment Diagrams



Component Diagram

- What is the Component Diagram?

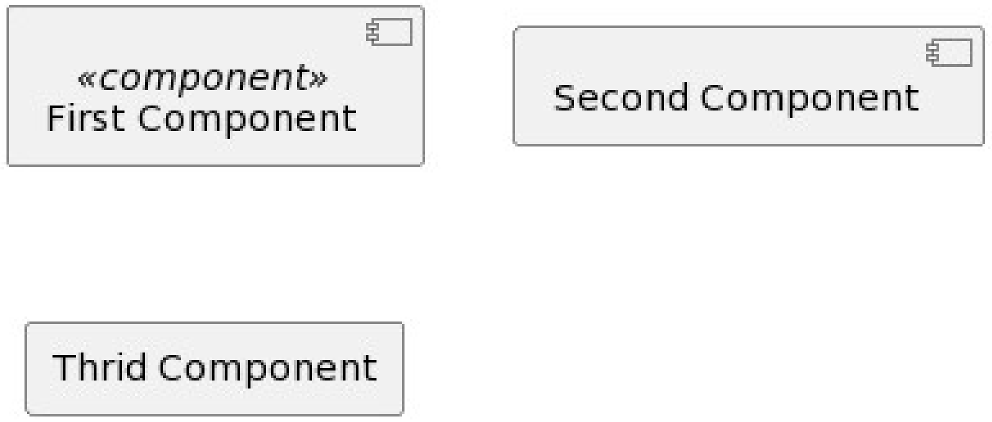
  - **Component Diagram:** divides a complex system into multiple components and shows the inter-relationships between the components.

  - The term '**component**': a module of classes that represents independent system or subsystem with the ability to interface with the rest of a more complex system.

    – Component diagram is useful to:
      - Show the system's physical structure (organization of the system!).
      - Show the system's static components and their relations.

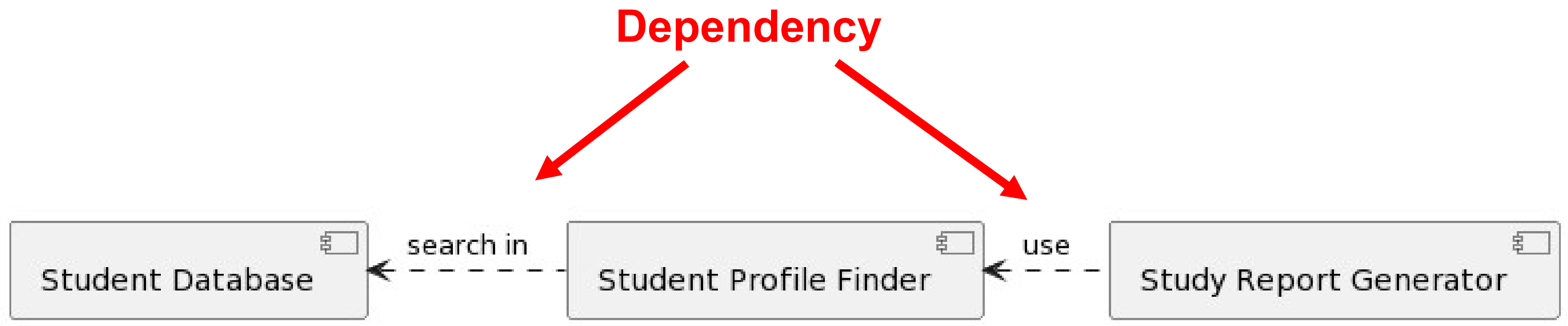
Component Diagram (cont.)

- Common elements in the diagram:
  - **Component:** represents a modular part of a system that encapsulates its contents. It can be represented by different ways:
    - **A rectangle with the stereotype <<component>> and/or icon.**
    - A rectangle with the component icon.
    - A rectangle with the name of the component.



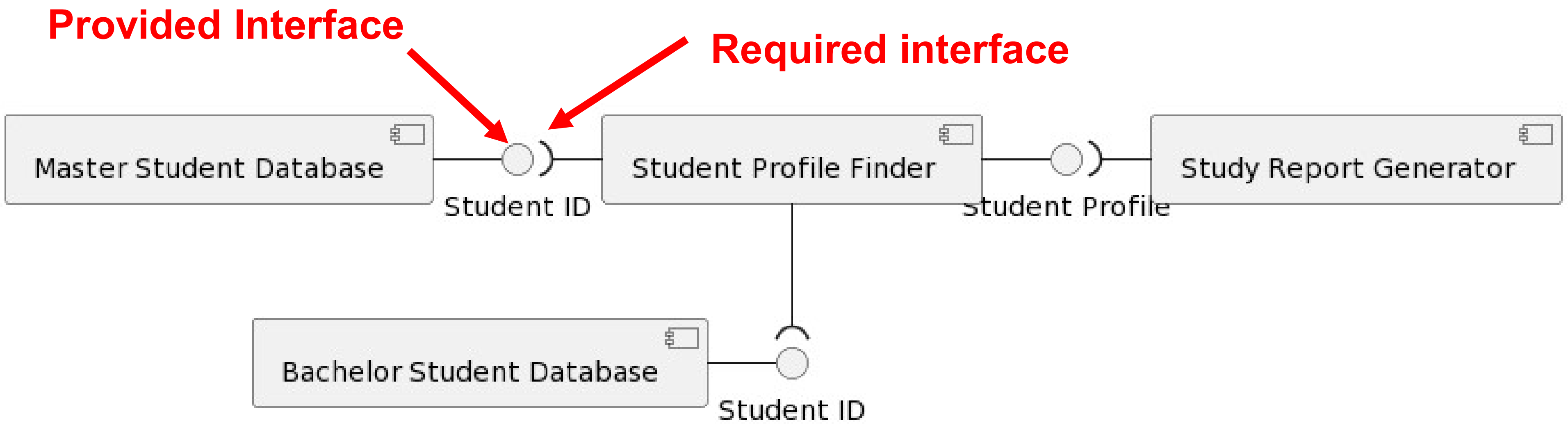
Component Diagram (cont.)

- Common elements in the Component Diagram:
  - **Dependency:**
    - Indicates that the functioning of one element depends on the existence of another element. (Thinking about the *#include* statement)



Component Diagram (cont.)

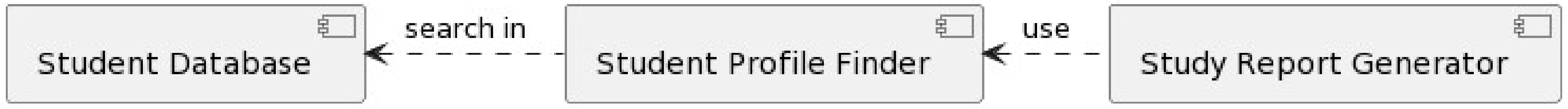
- Common elements in the Component Diagram:
  - **Assembly:**
    - **Provided interface**: symbols with a complete circle at the end represent an interface
    - **Required interface**: symbols with a half circle at the end represent an interface that the component requires.



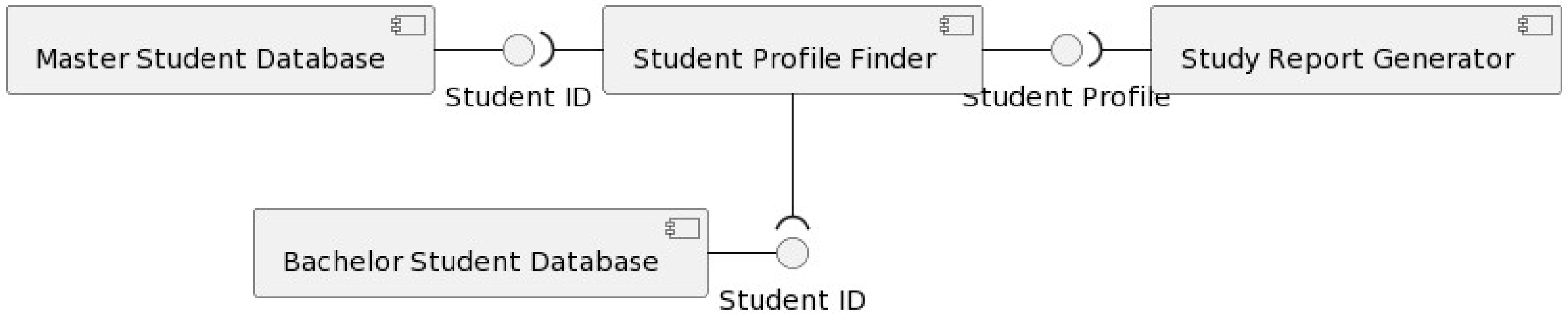
Component Diagram (cont.)

- **Discussion**:
  - In the following two diagrams, what could be the difference?

Dependency



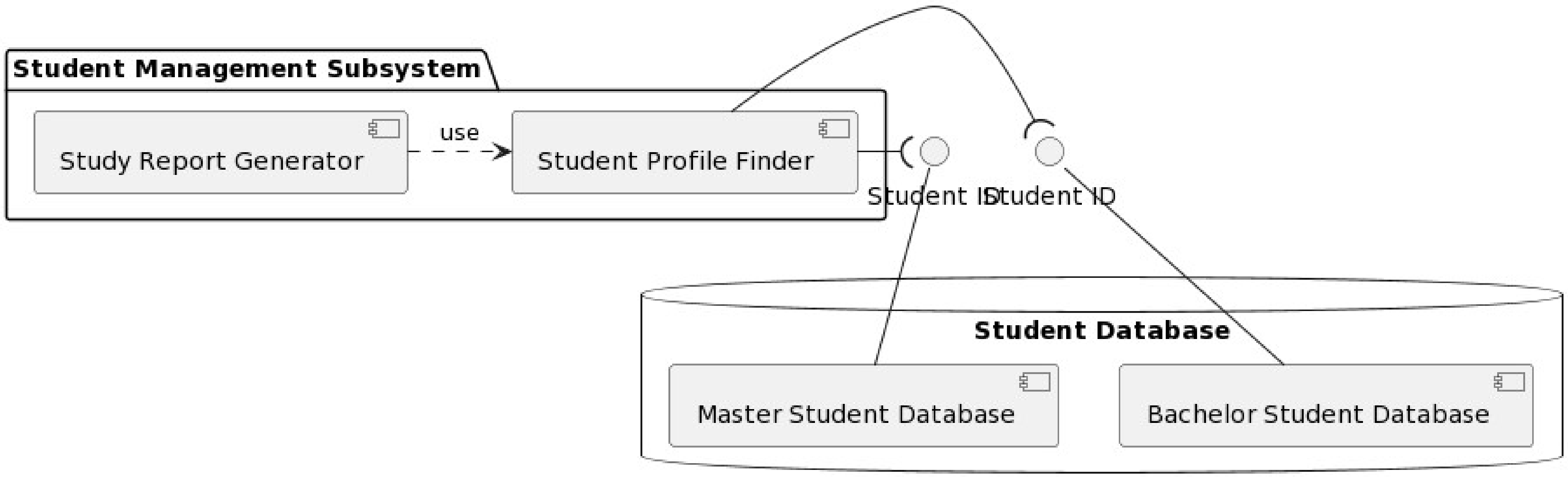
Assembly



- Dependency between two components on the classifier level expresses a potential assembly relationship between the two corresponding instances in system run-time.
- They are modeling the system at different abstraction

Component Diagram (cont.)

- Common elements in the Component Diagram:
  - **Group and package:**



Deployment Diagram

- What is the Deployment Diagram?

  ■ **Deployment Diagram:** a type of structural diagram that shows a system’s physical layout, revealing which pieces of software run on what pieces of hardware.

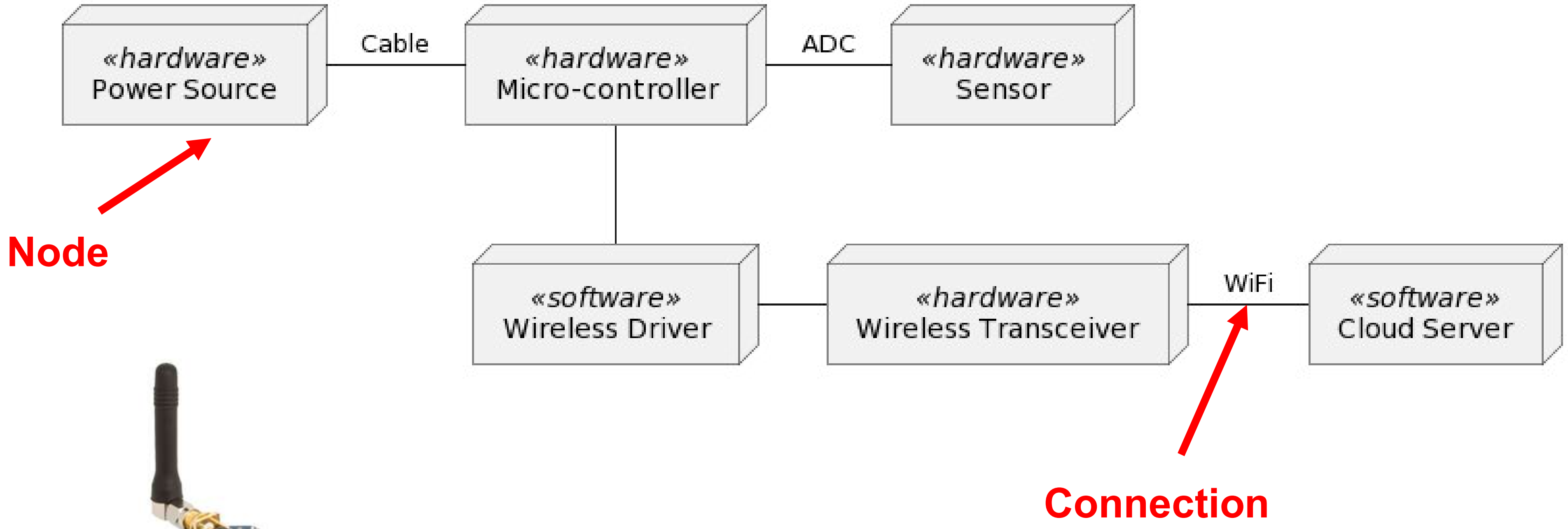
  – It shows the physical deployment of the software elements.

  – It illustrates the runtime processing for hardware.

  – It provides the topology of the hardware system.

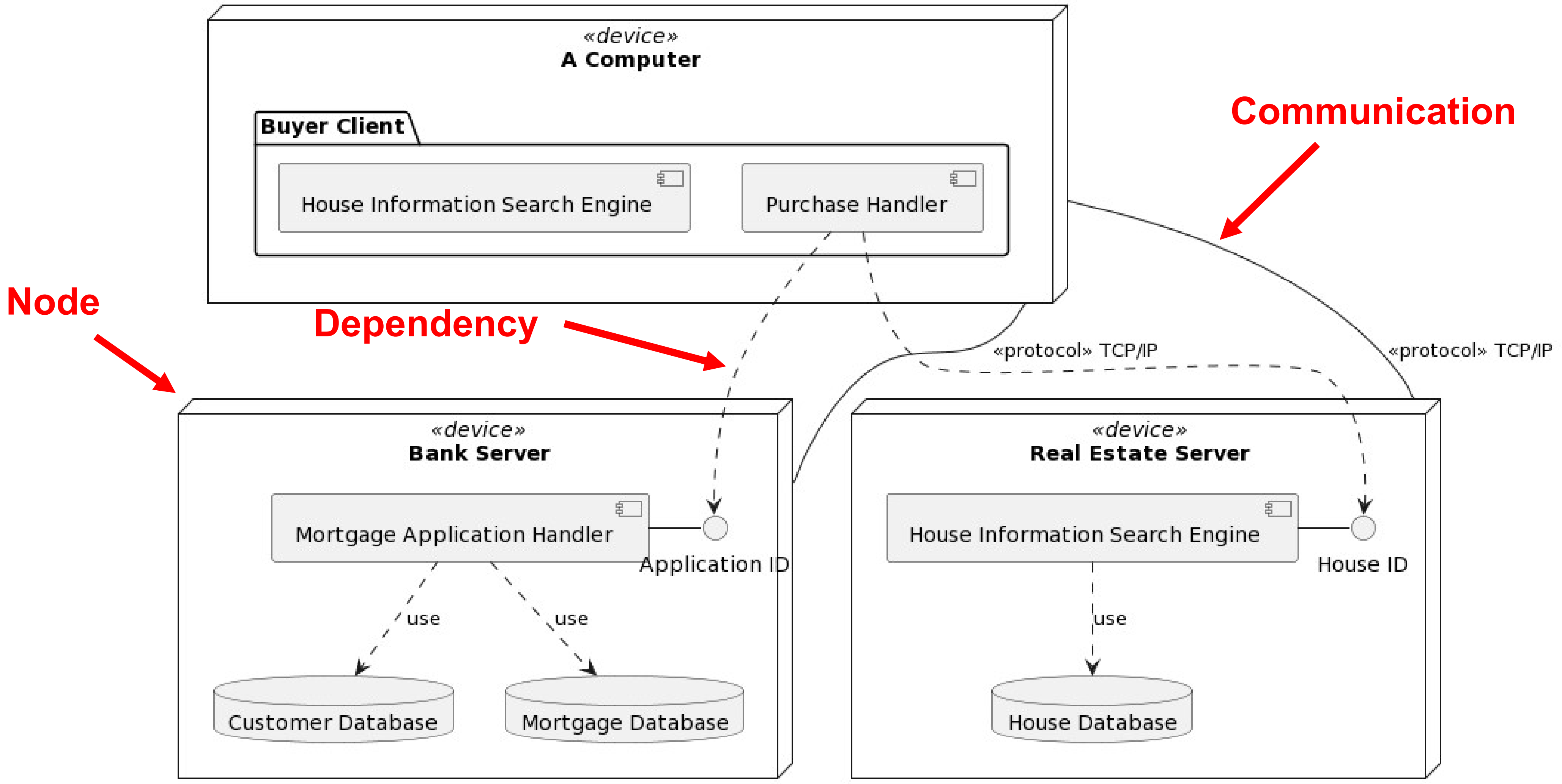
Deployment Diagram (cont.)

- Modeling a wireless sensing system:

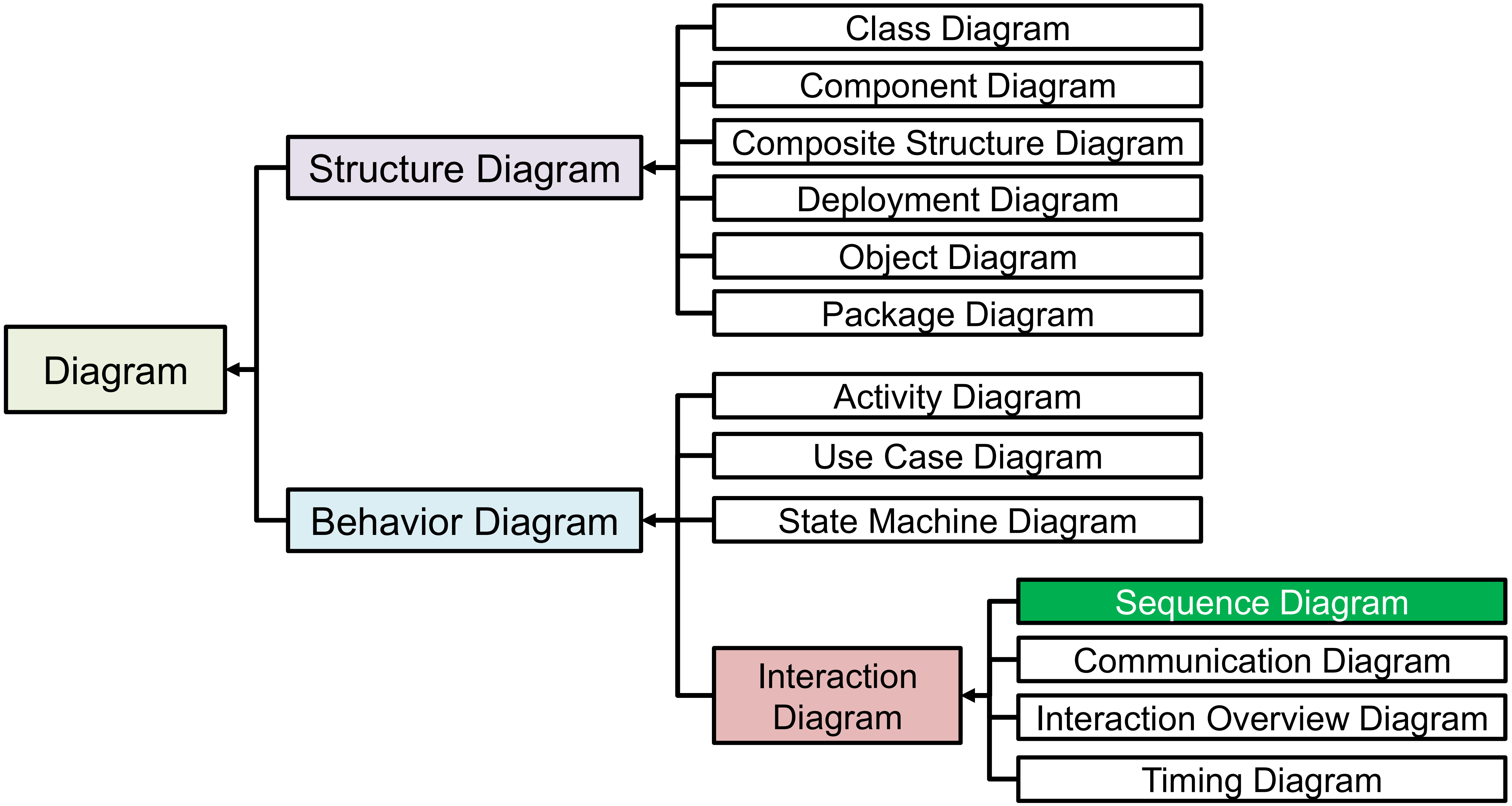


Deployment Diagram (cont.)

- Another example:



Sequence Diagram



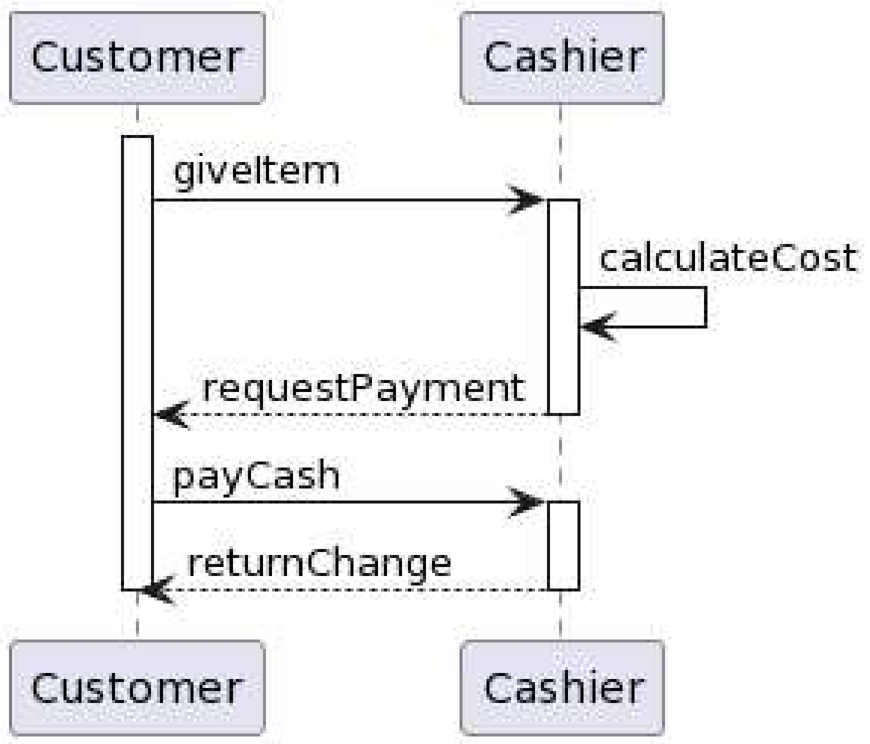
Sequence Diagram (cont.)

- What is the Sequence Diagram?

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

Sequence Diagram (cont.)

- **Discussion**:
  - What do you see in this diagram?
  - What are the elements in this diagram?
  - What message(s) this diagram may try to deliver?



Think → Pair → Share

Sequence Diagram (cont.)

- What is the Sequence Diagram?

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

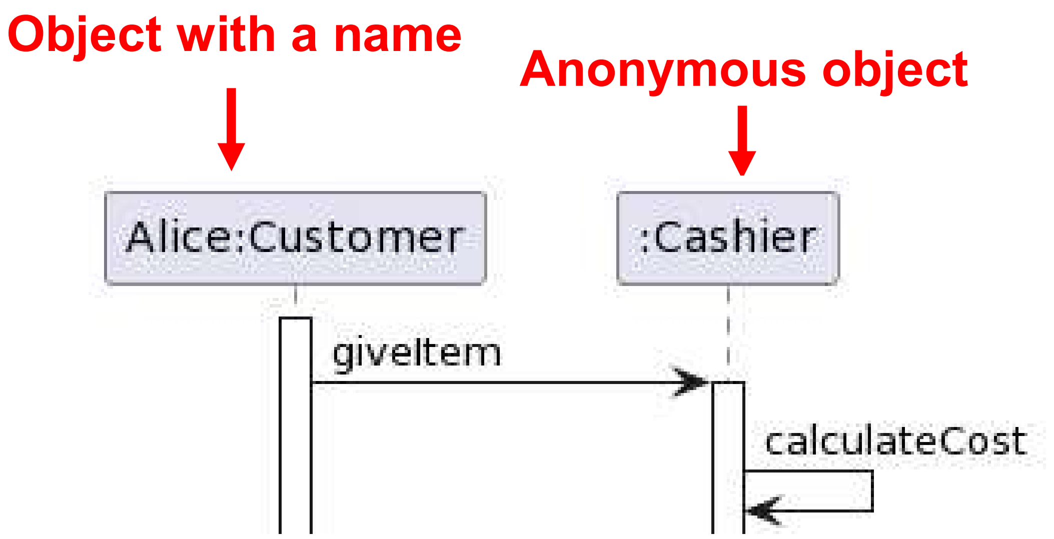
- It is a behavioral diagram that shows:
  - Lifelines of participants
  - Messages shared
  - How objects are activated
  - Which object is controlling the flow
- Does not provide a lot of implementation details.

Sequence Diagram (cont.)

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

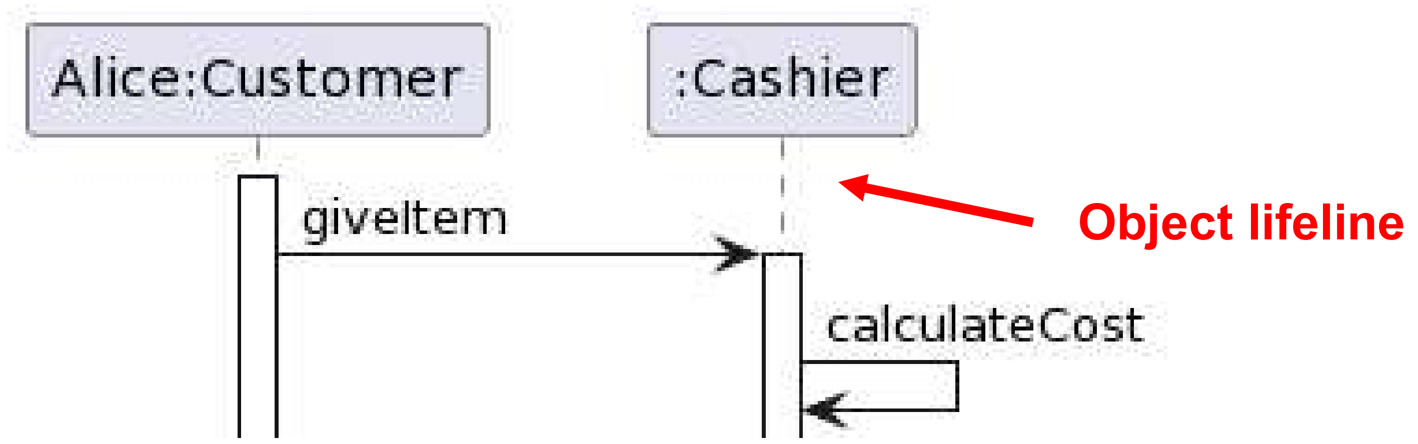
Name syntax: <objectname>:<classname>

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



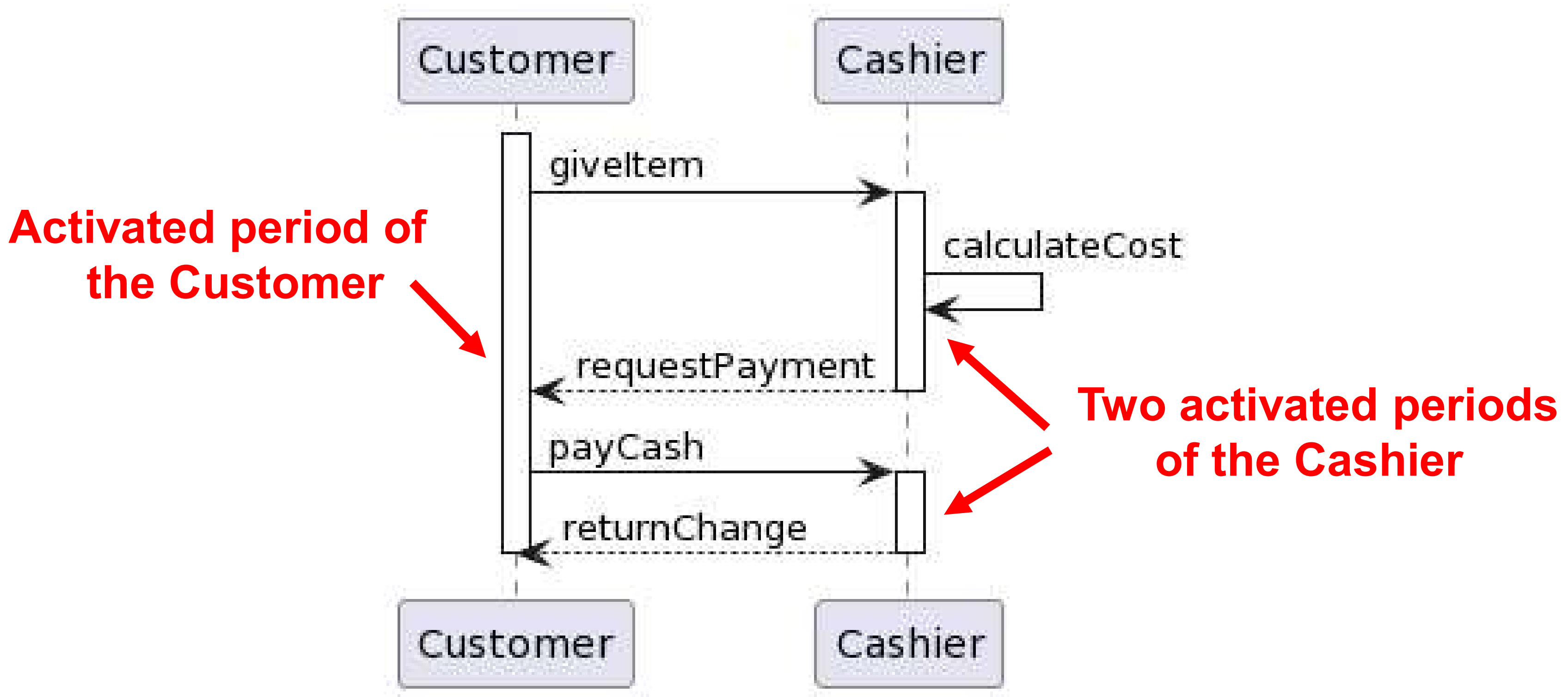
Sequence Diagram (cont.)

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



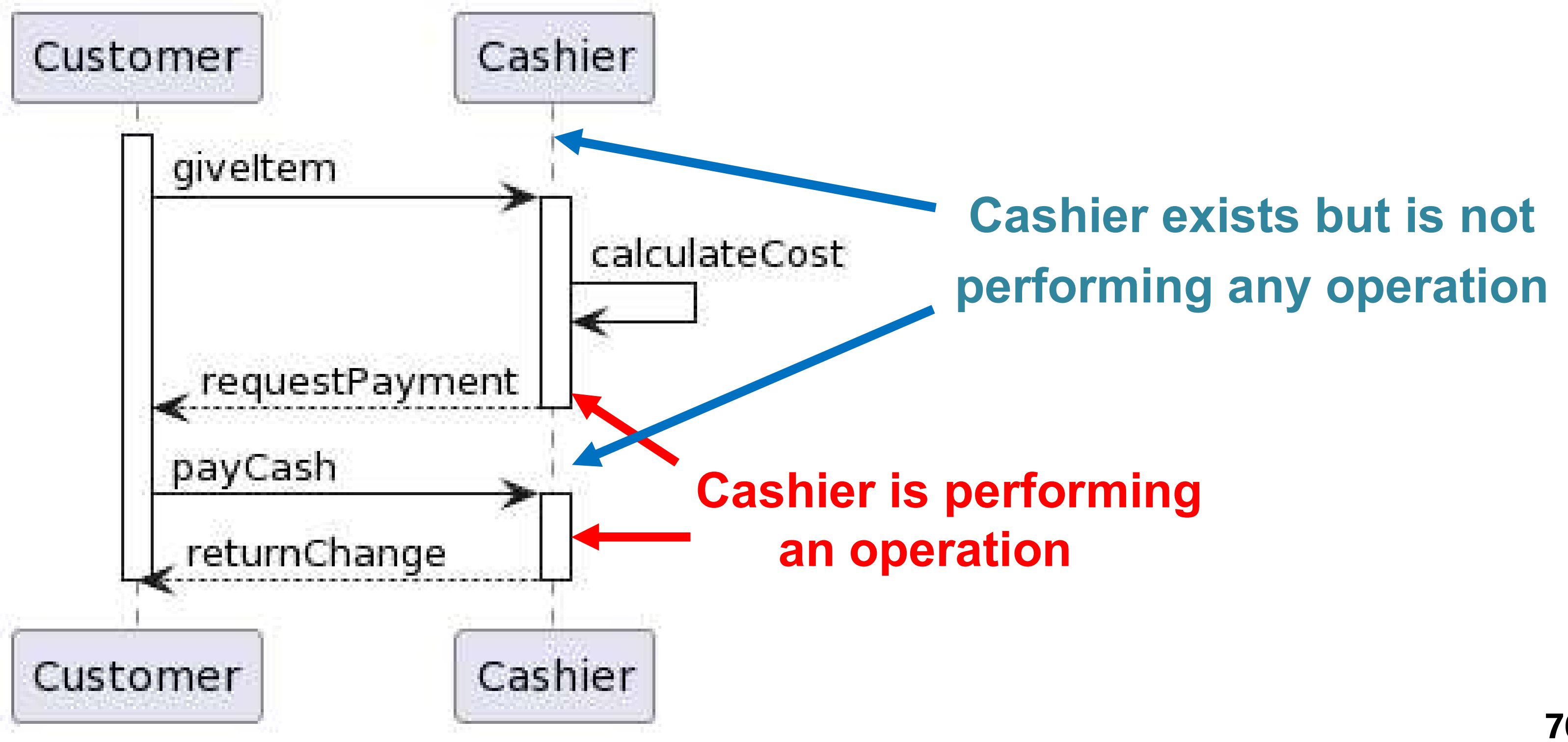
Sequence Diagram (cont.)

- 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/action (e.g., running its code or waiting for another participant's method to finish).



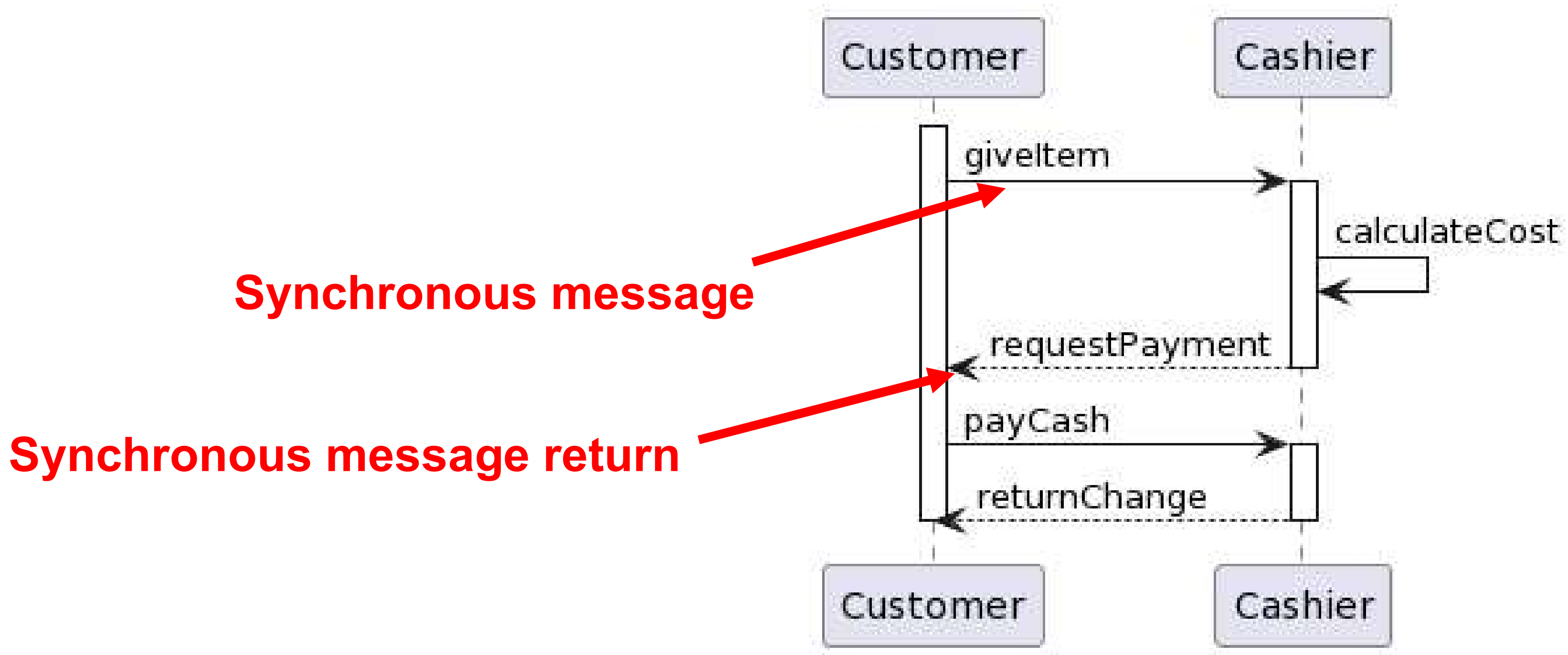
Sequence Diagram (cont.)

- Difference between activation and lifeline?
  - **Activation**: a thin rectangle on the lifeline that represents the period during which a participant is performing an operation/action (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.)

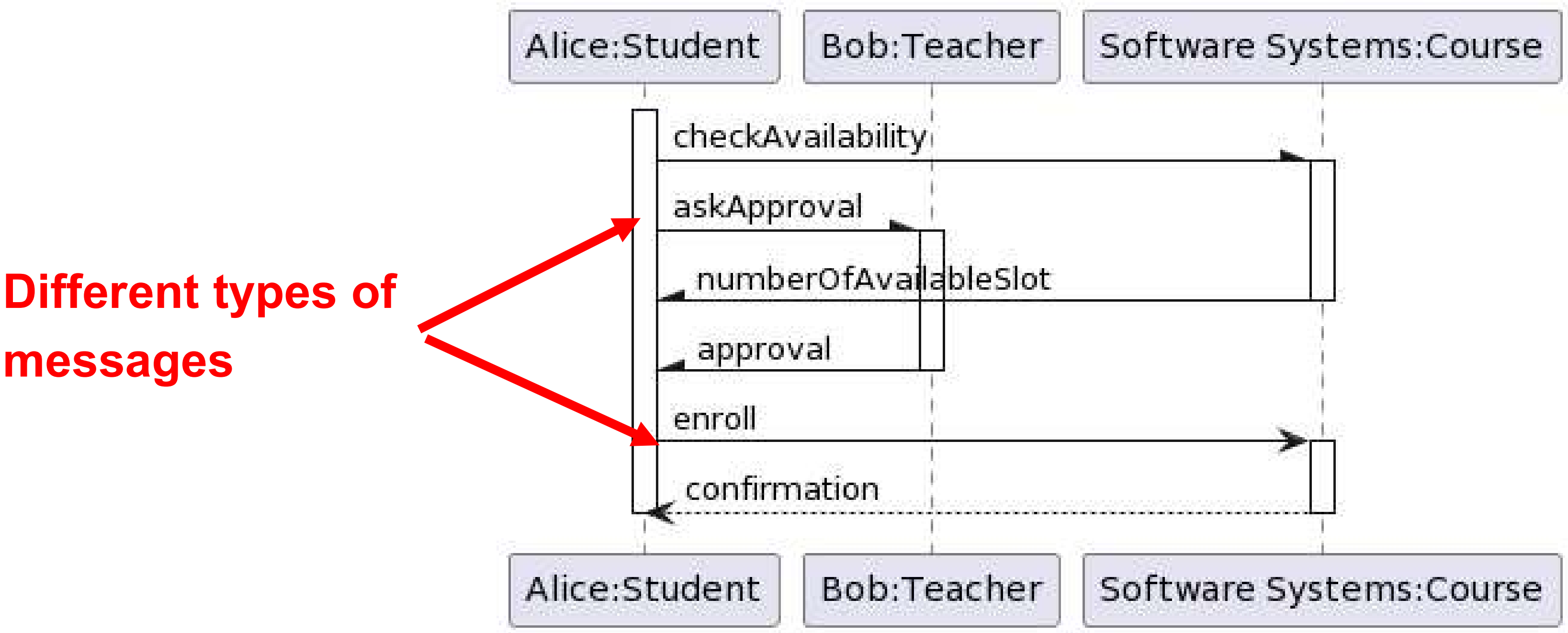
- Common elements in a sequence diagram:
  - **Message (method call)**: communication between participants.
    - Synchronous message and return.
      - If the caller sends a synchronous message, it must wait until it receives a response (message return) from the target.



Sequence Diagram (cont.)

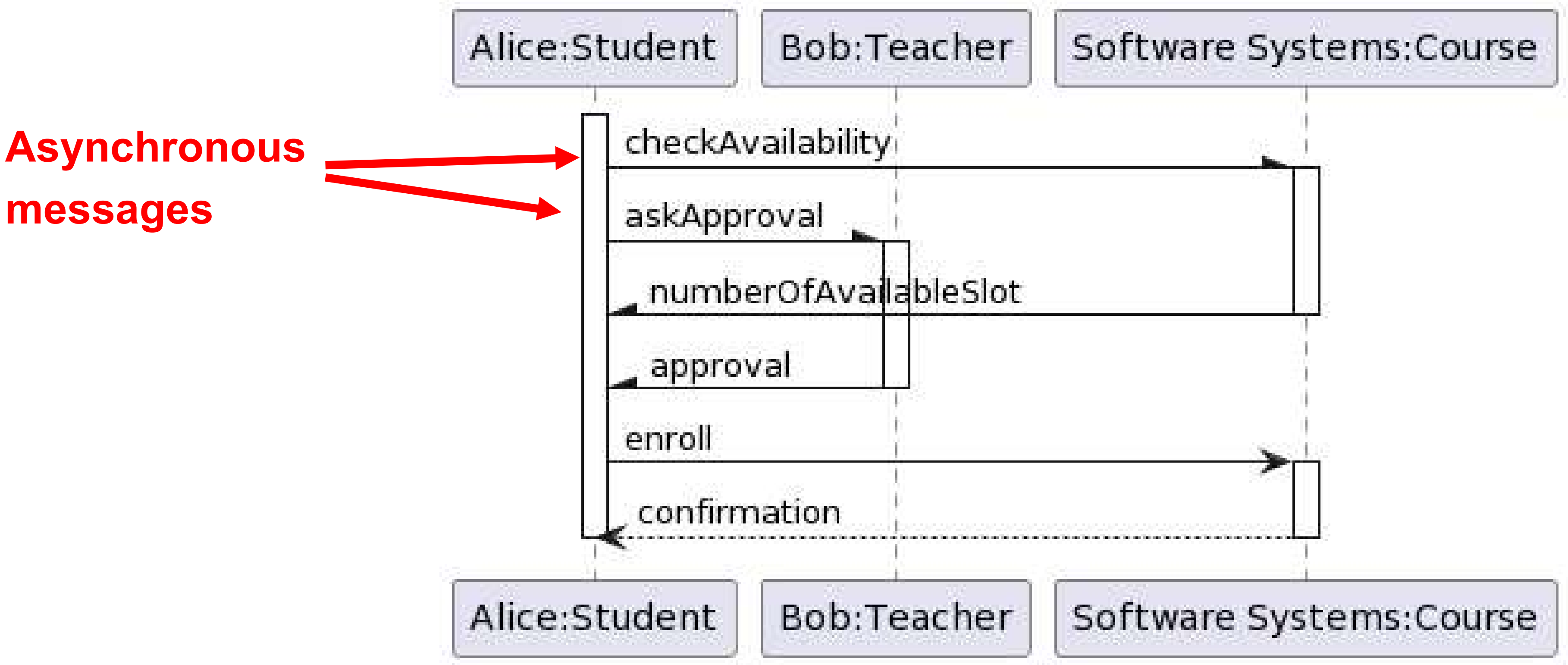
- Common elements in a sequence diagram:

  - **Discussion**:
    - In the following diagram, you can see a type of messages that is different from the synchronous messages. What could this type of message mean? What do they represent?



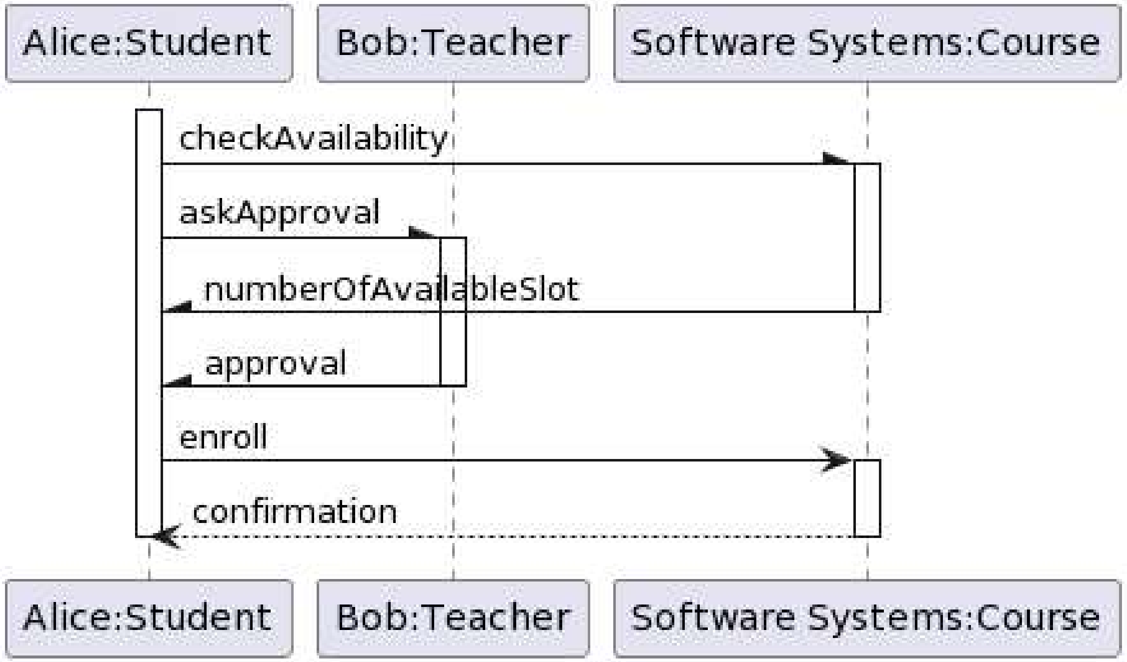
Sequence Diagram (cont.)

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



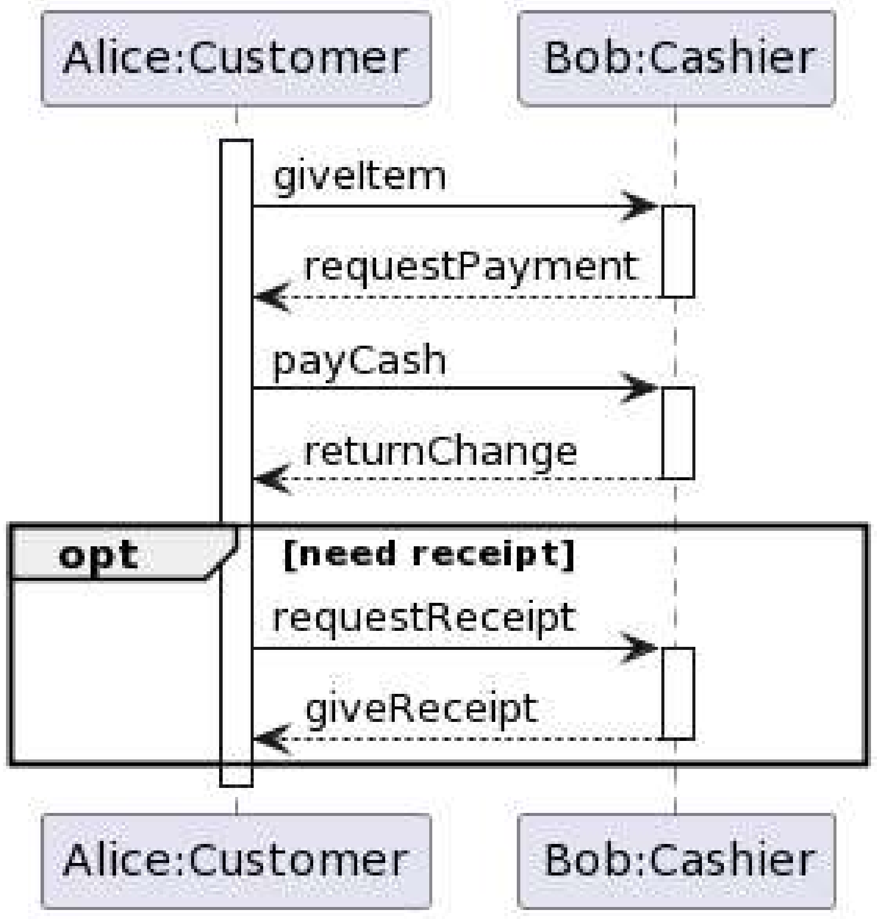
Sequence Diagram (cont.)

- Common elements in a sequence diagram:
  - **Message (method call)**: communication between participants.
  - The key difference lies in the timing and waiting behavior:
    - Synchronous: involve immediate and direct interaction (the sender is waiting!)
    - Asynchronous: involve non-blocking communication. The sender can continue its execution without waiting for a reply.



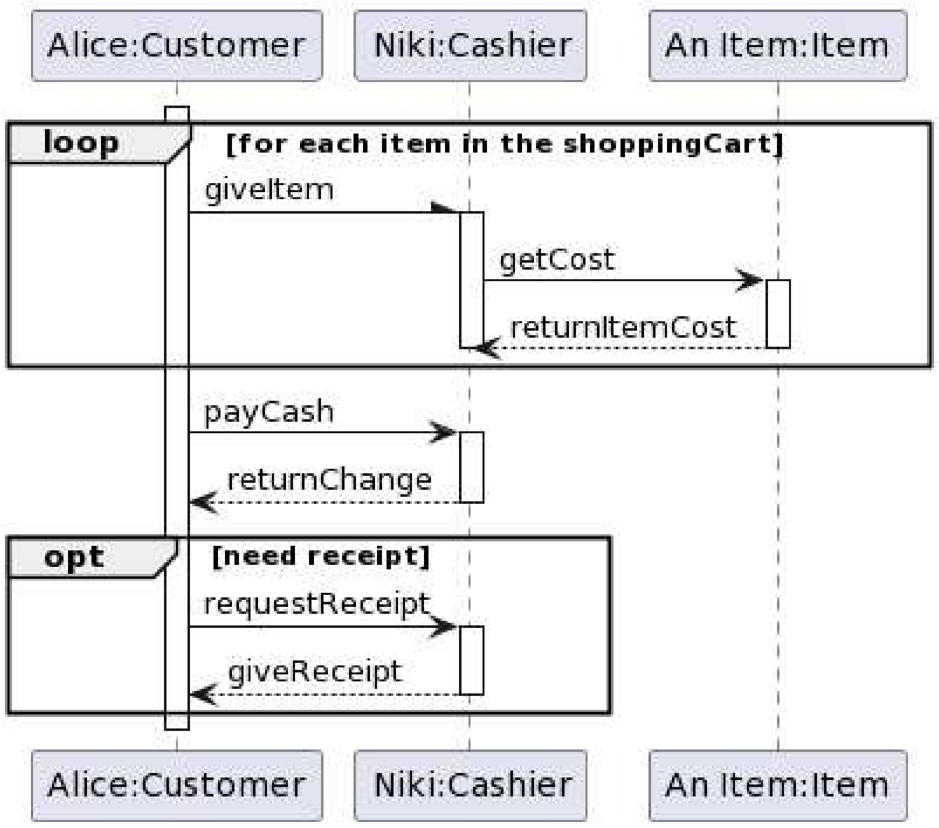
Sequence Diagram (cont.)

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



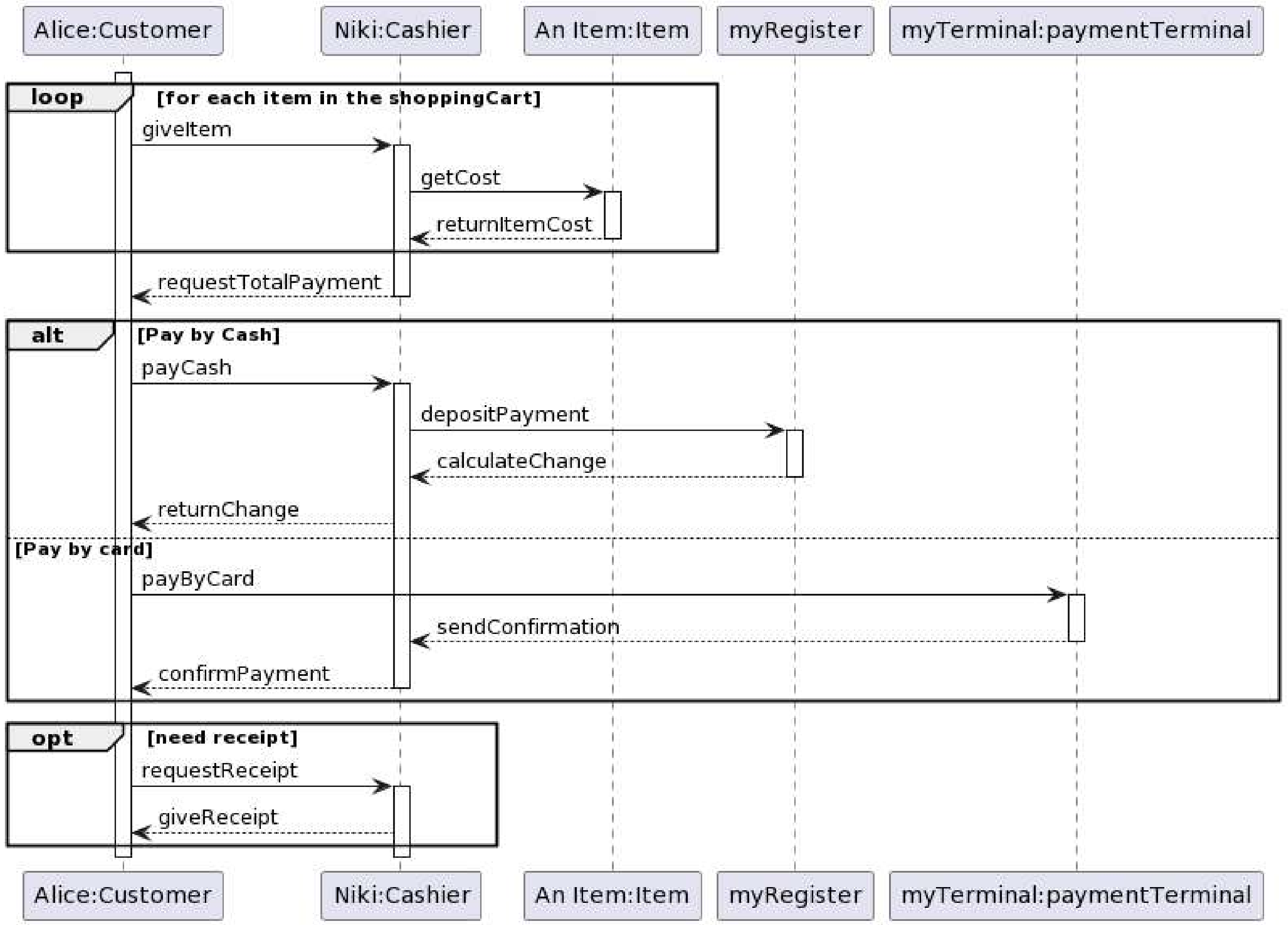
Sequence Diagram (cont.)

- 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 Component, Class and Sequence diagrams:
  - Work on the modeling assignment.