CHAPTER FIVE

Object Oriented Requirement Analysis

AN OVERVIEW OF ANALYSIS

- ❑Object-oriented analysis examines system requirements from the perspective of classes and objects found in the specified domain.
- ❑ Object-oriented analysis looks at the real-world environment in which a system will operate, with this environment consisting of people and things interacting to create some result.
- ❑The people and things are first analyzed in the most abstract form and these abstractions become the class.
- ❑The abstraction is analyzed and reanalyzed in multiple iterations until all objects are uniquely identified.
- ❑Object characteristics and their behaviors are then analyzed to establish the various states an object can have and to define the methods the object will use to create action.

ANALYSIS CONCEPTS

- Entity, Boundary, and Control Objects
  - ✓ Entity, Boundary, Control taxonomy is similar to that of the MVC architecture where Entity = Model, Control = Controller, Boundary = View.
  - ✓ **Entities:** Objects representing system data, often from the domain model.
  - ✓ **Boundaries:** Objects that interface with system actors
    - e.g. a user or external service. Windows, screens and menus are examples of boundaries that interface with users.
  - ✓ **Controls:** Objects that mediate between boundaries and entities.
    - These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.

- Communication allowed between these classes are shown below

	Entity	Boundary	Control
Entity	X		X
Boundary			X
control	X	X	X

CLASS DIAGRAMS

- A class diagram is used to show the existence of classes and their relationships in the logical view of a system.
- A single class diagram represents a view of the class structure of a system.
- During analysis, we use class diagrams to indicate the common roles and responsibilities of the entities that provide the system’s behavior.
- During design, we use class diagrams to capture the structure of the classes that form the system’s architecture.
- The two essential elements of a class diagram are classes and their basic relationships.

THE CLASS NOTATION

- ❑The class icon consists of three compartments, with the first occupied by the class name, the second by the attributes, and the third by the operations.
- ❑A name is required for each class and must be unique to its enclosing namespace.
  - ✓By convention, the name begins in capital letters, and the space between multiple words is omitted.
  - ✓Again by convention, the first letter of the attribute and operation names is lowercase, with subsequent words starting in uppercase, and spaces are omitted just as in the class name.
  - ✓Since the class is the namespace for its attributes and operations, an attribute name must be unambiguous in the context of the class.
  - ✓So must an operation name, according to the rules in the chosen implementation language.

- The format of the attribute and operation specifications is shown here:
- Attribute specification format:

  visibility attributeName : Type [multiplicity] = DefaultValue {property string}

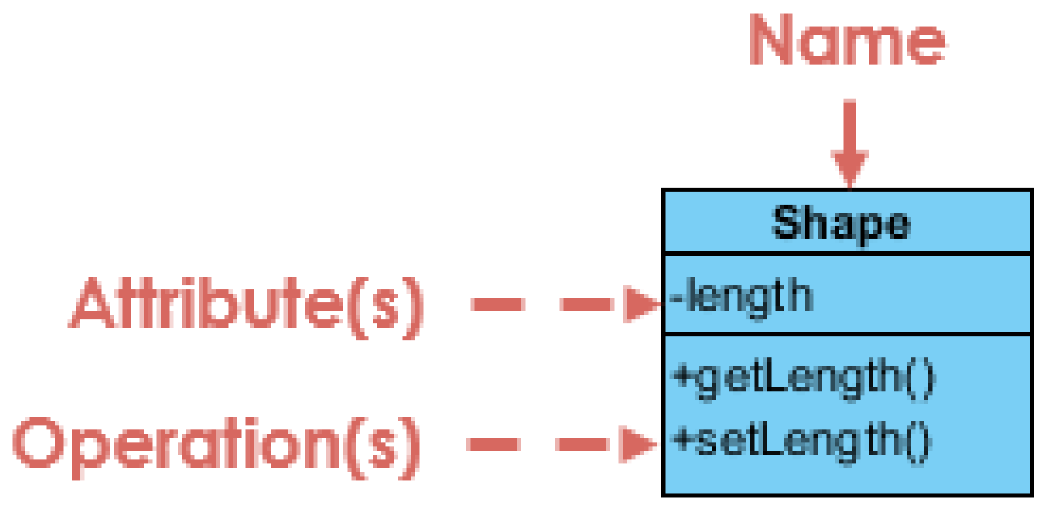
- Operation specification format:

  visibility operationName (parameterName : Type) : ReturnType {property string}

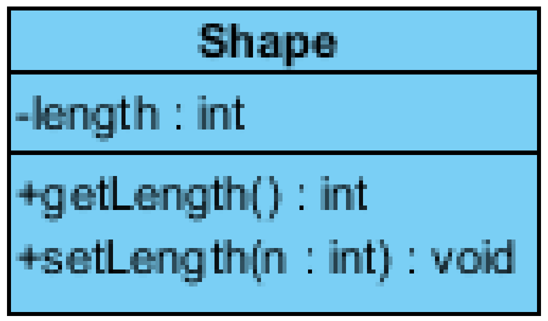
VISIBILITY

- ❑ The following visibility can be applied to class associations, attributes, and operations.
- ❑ We may specify visibility by adorning the appropriate element with the following symbols:
  - ✓ Public (+) : Visible to any element that can see the class
  - ✓ Protected (#): Visible to other elements within the class and to subclasses
  - ✓ Private (-): Visible to other elements within the class
  - ✓ Package (~): Visible to elements within the same package

EXAMPLE OF CLASS NOTATION



Class without signature

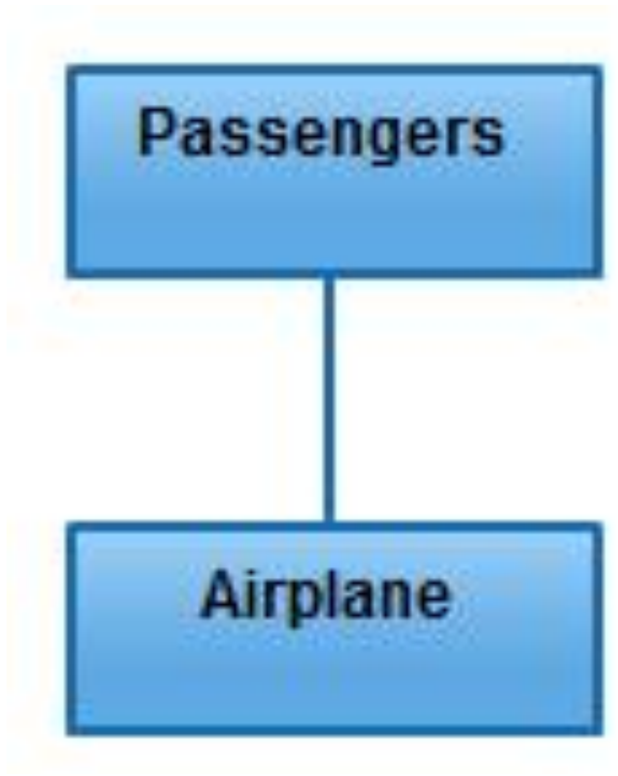


Class with signature

- Relationship
  - Classes rarely stand alone; instead, they collaborate with other classes in a variety of ways.
  - Classes are interrelated to each other in specific ways.
  - In particular, relationships in class diagrams include different types of logical connections.
  - The following are such types of logical connections that are possible in UML:
    - Association
    - Directed Association
    - Reflexive Association
    - Multiplicity
    - Aggregation
    - Composition
    - Inheritance/Generalization
    - Realization

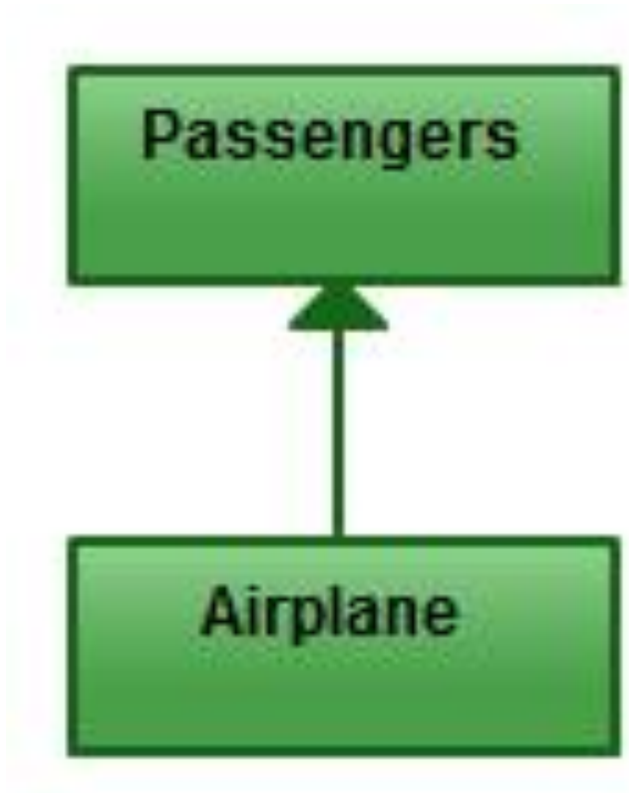
❑Association

- ✓is a broad term that encompasses just about any logical connection or relationship between classes.
- ✓For example, passenger and airline may be linked as below:



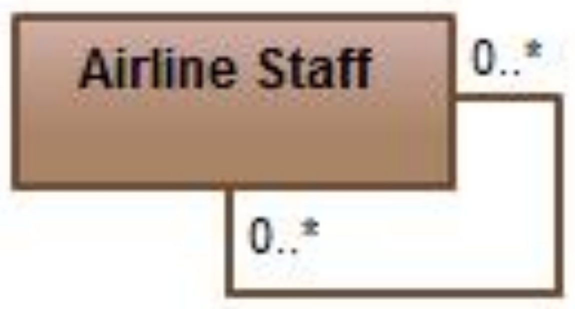
Directed Association

- refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

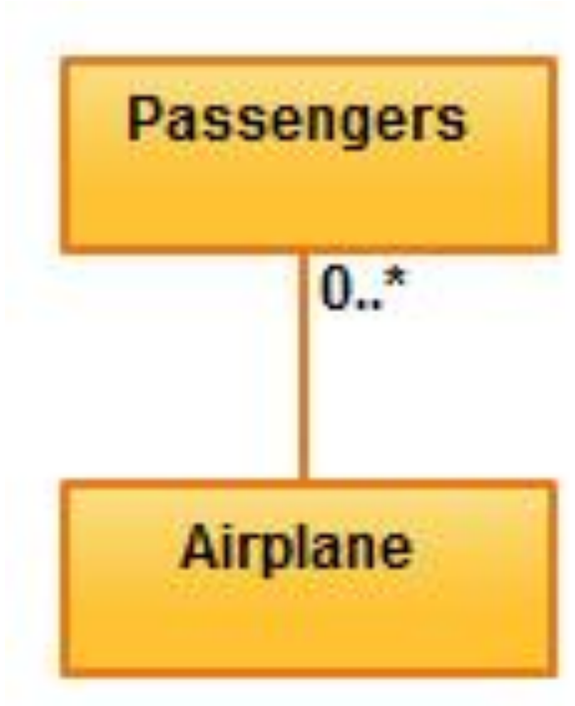


Reflexive Association

- ✓ This occurs when a class may have multiple functions or responsibilities.
  - For example, a staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member. If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.



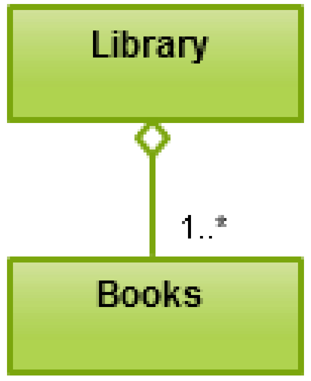
- Multiplicity
  - ✓ is the active logical association when the cardinality of a class in relation to another is being depicted.
    - For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means "zero to many".



- Some typical examples of multiplicity:

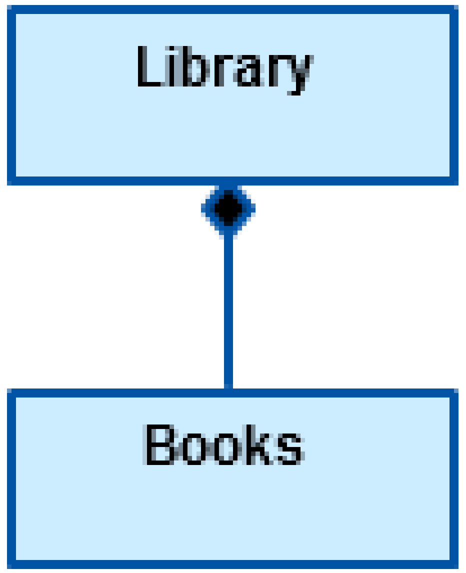
Multiplicity	Option	Cardinality
0..0	0	Collection must be empty
0..1		No instances or one instance
1..1	1	Exactly one instance
0..*	*	Zero or more instances
1..*		At least one instance
5..5	5	Exactly 5 instances
m..n		At least m but no more than n instances

- Aggregation
  - refers to the formation of a particular class as a result of one class being aggregated or built as a collection.
    - For example, the class "library" is made up of one or more books, among other materials.
  - In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved.
  - To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

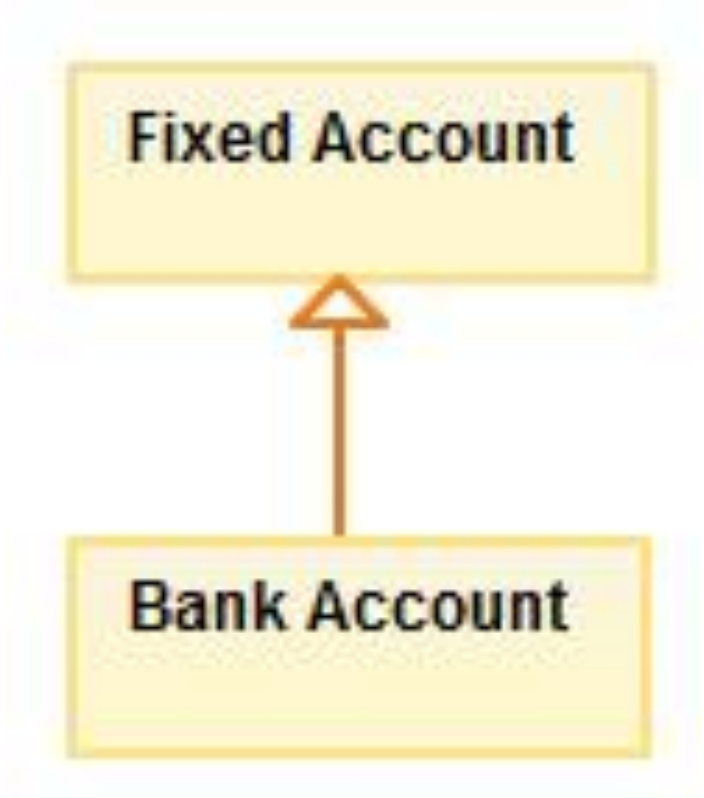


Composition

- The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class.
- That is, the contained class will be obliterated when the container class is destroyed.
- To show a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

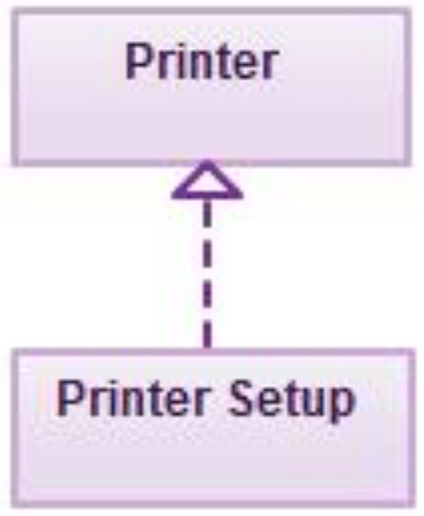


- Inheritance / Generalization
  - refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class.
  - In other words, the child class is a specific type of the parent class.
  - To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.



Realization

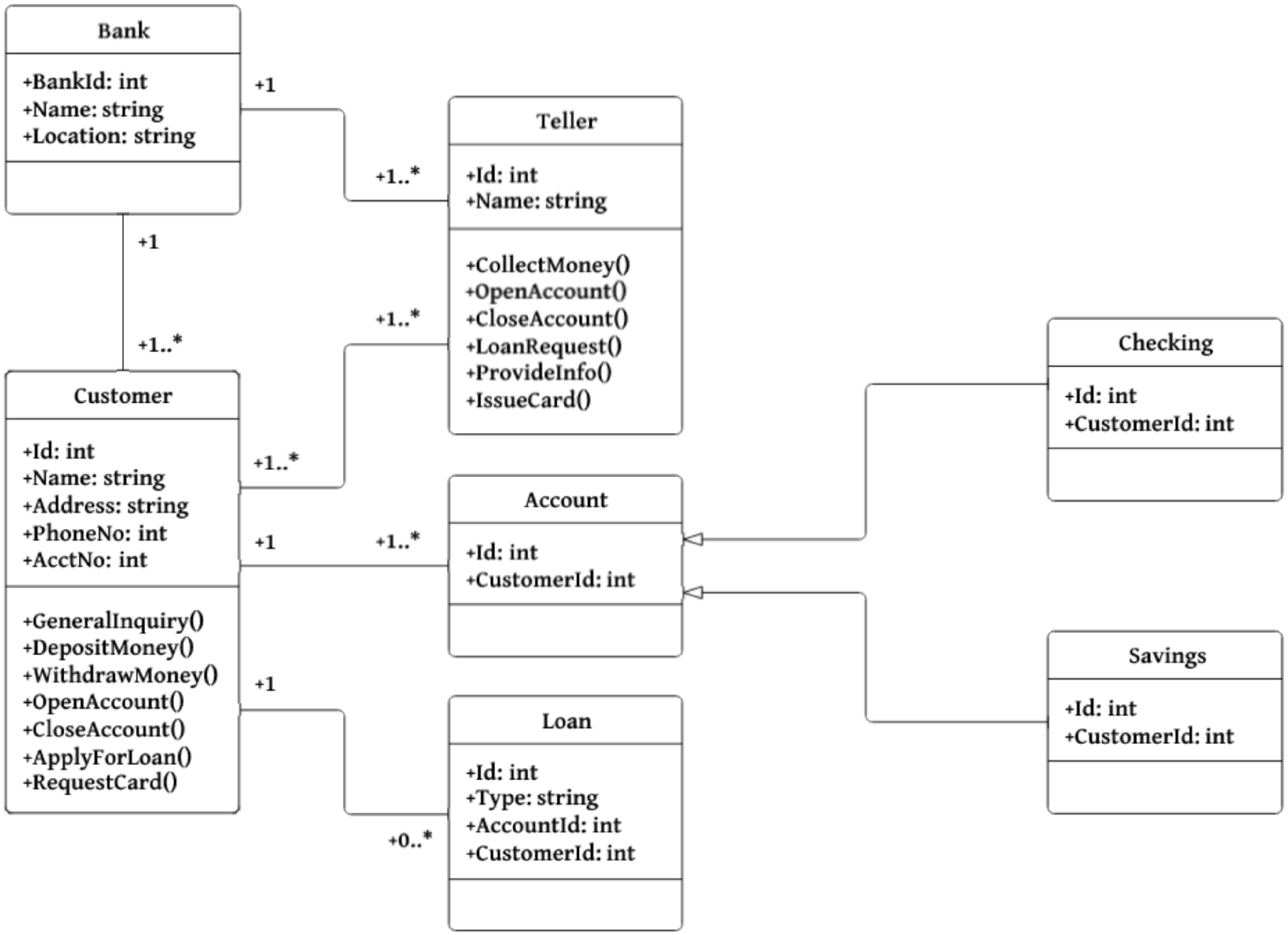
- ✓ denotes the implementation of the functionality defined in one class by another class.
- ✓ To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class to the class that implements the function.
- ✓ In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.



FROM USE CASES TO CLASSES

1. Identify **classes** of objects in the problem description
   - Heuristic: nouns may be classes
   - But: verbs can be nominalized (iterate->iteration, withdraw->withdrawal)
   - Heuristic: does it store state (does it have attributes) and does it have operations?
2. Identify attributes of each class:
   - What characteristics are important to keep track of?
   - What data must be maintained or remembered during object's life?
3. Identify methods of each class:
   - What actions must the object be able to perform?
   - What actions can be done to the object (e.g., to change its state)?
   - Use cases may correspond to methods
4. Identify parameters and return types of methods:
   - What input does a method need from caller? These are parameters.
   - Does the method generate a result value?
   - The return type must match the data type of data to be returned.

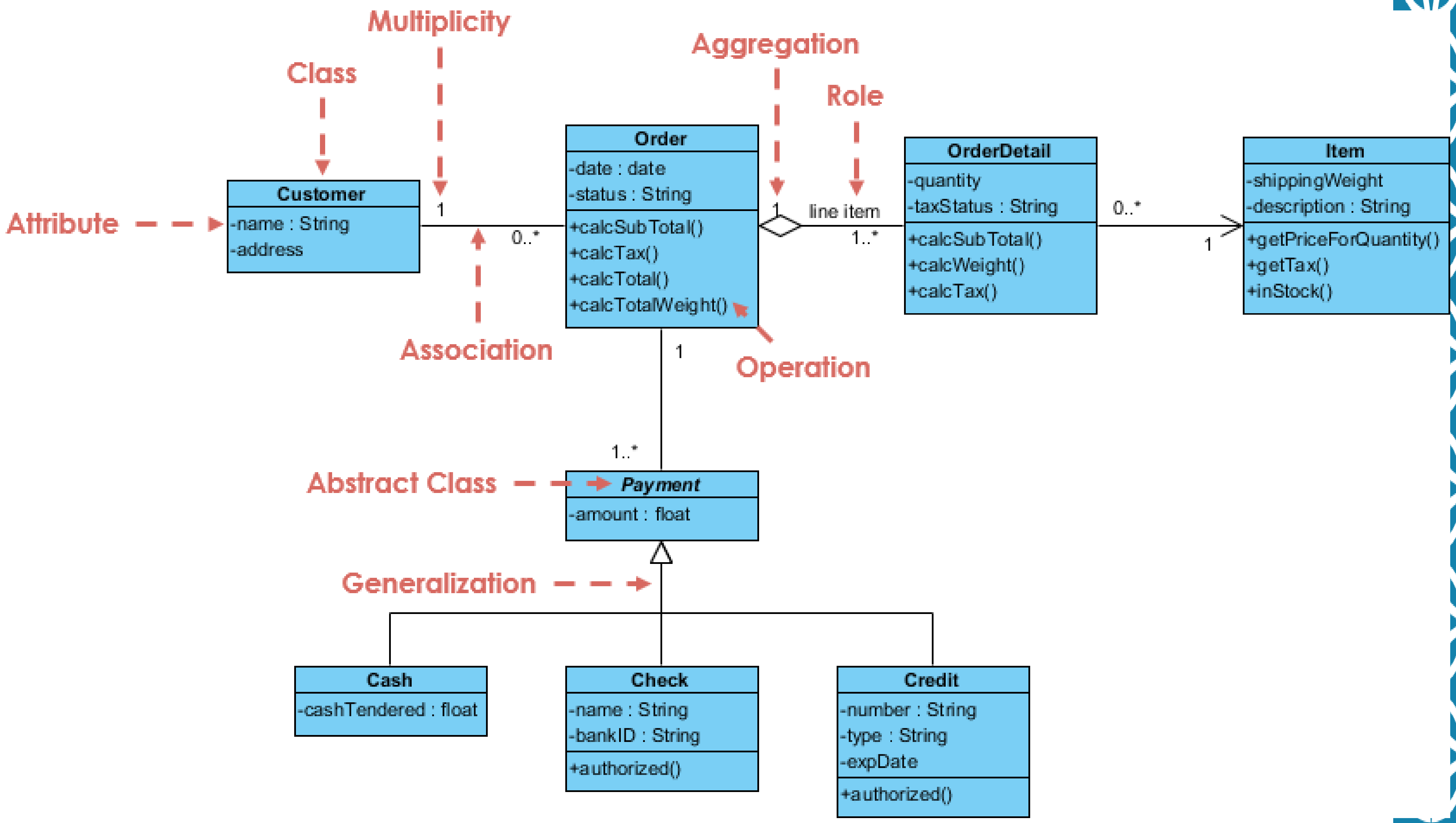
CLASS DIAGRAM EXAMPLE 1: BANKING SYSTEM



EXERCISE

- Order management system

CLASS DIAGRAM EXAMPLE 2: ORDER SYSTEM



Modeling Interactions between Objects: Sequence Diagrams

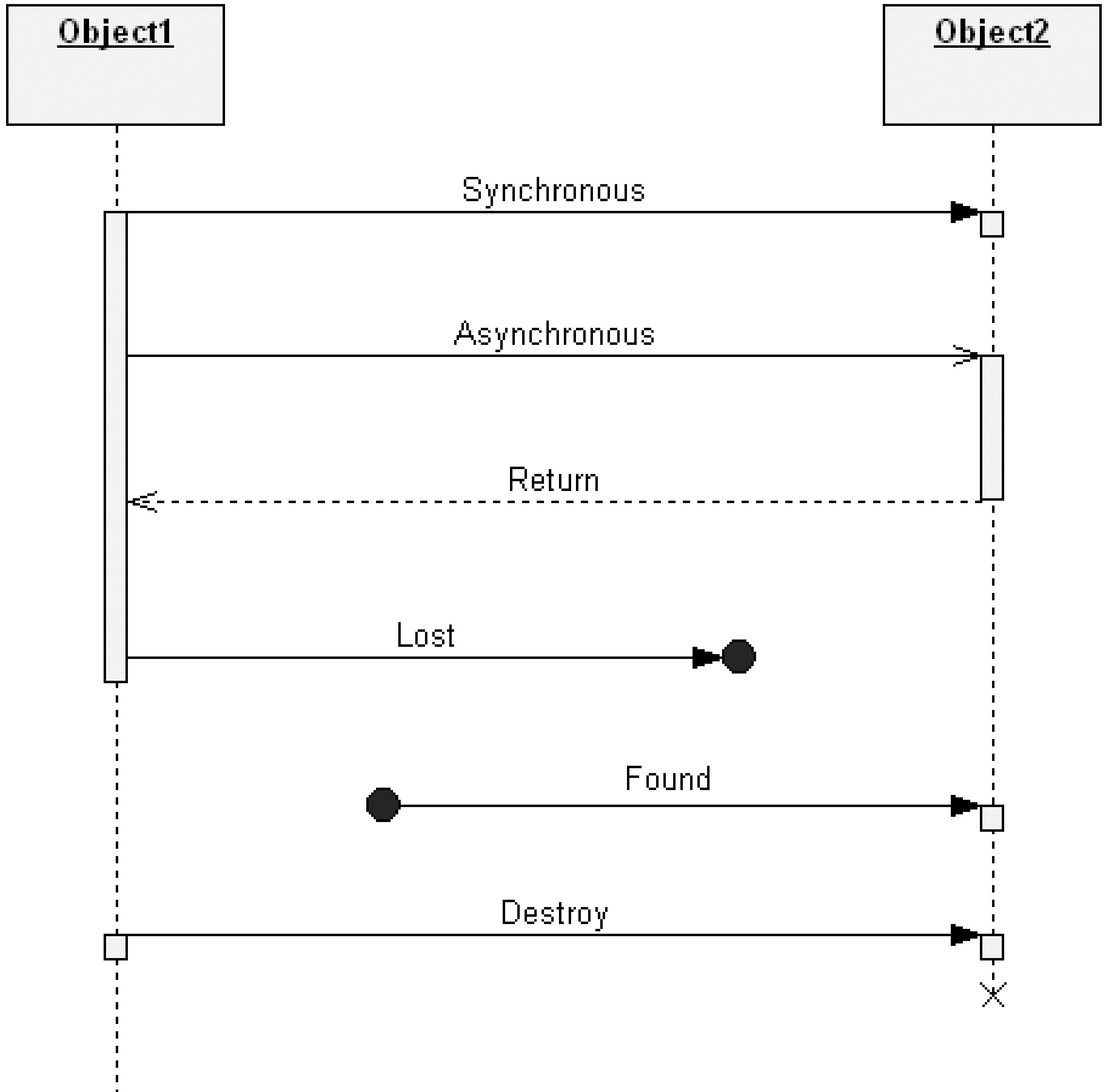
- A sequence diagram is used to trace the execution of a scenario in the same context as a communication diagram.
  - A sequence diagram is simply another way to represent a communication diagram.
- A sequence diagram duplicates most of the semantics of the communication diagram.
  - The advantage of using a sequence diagram is that it is easier to read the passing of messages in relative order.

LIFELINES AND MESSAGES

- In sequence diagrams, the entities of interest are written horizontally across the top of the diagram.
- A dashed vertical line, called the *lifeline, is drawn below each object.*
  - *These indicate the existence of the object.*
- *Messages (which may denote events or the invocation of operations)* are *shown* horizontally.
- The endpoints of the message icons connect with the vertical lines that connect with the entities at the top of the diagram.
- Messages are drawn from the sender to the receiver.
- Ordering is indicated by vertical position, with the first message shown at the top of the diagram, and the last message shown at the bottom.
- As a result, sequence numbers aren't needed.

TYPES OF MESSAGES

- The notation used for messages (i.e., the line type and arrowhead type) indicates the type of message being used.

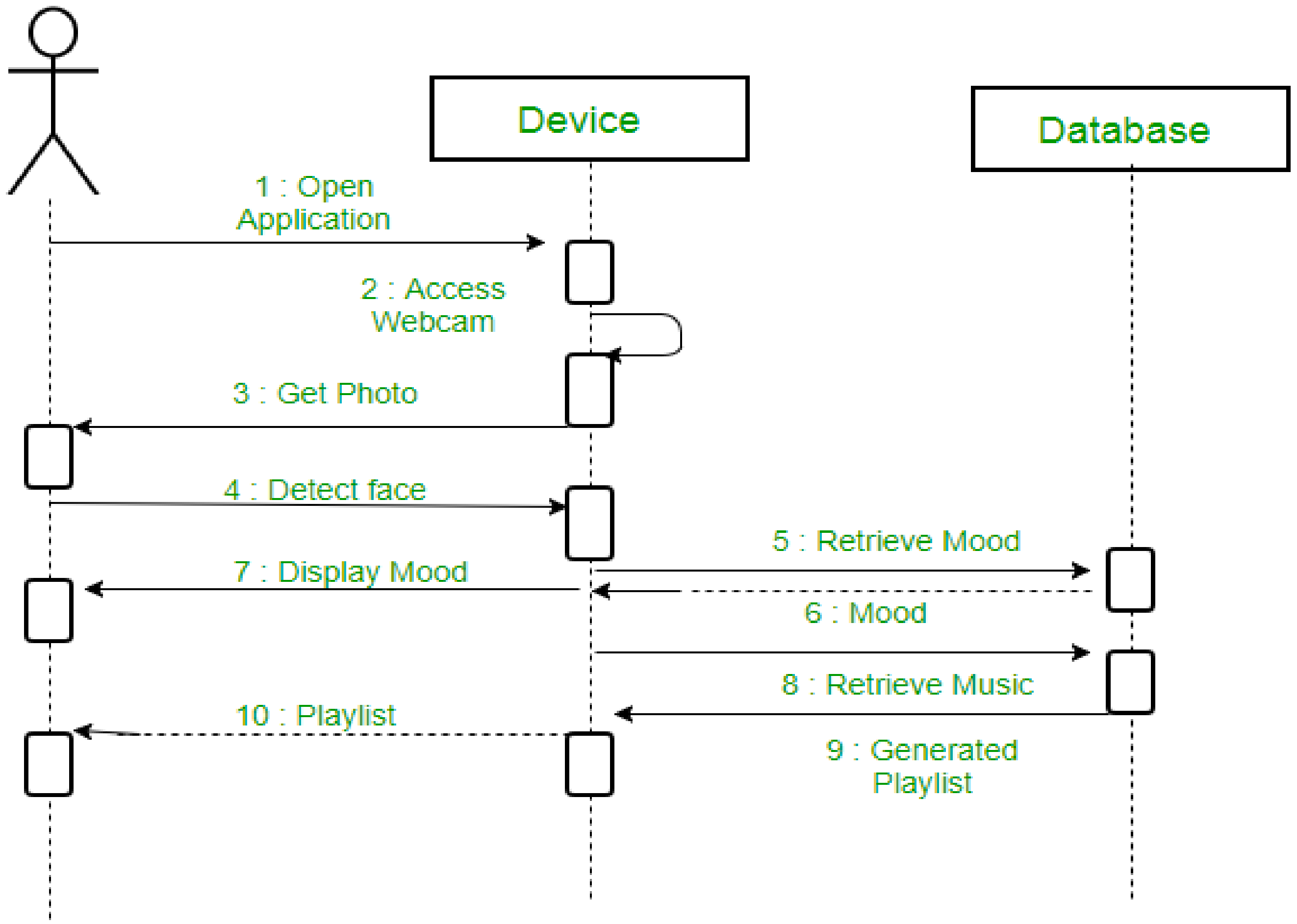


DESTRUCTION EVENTS

- A destruction event indicates when an object is destroyed.
- It is shown as an X at the end of a lifeline.
  - See the Object2 lifeline in the previous figure for example.

EXECUTION SPECIFICATION

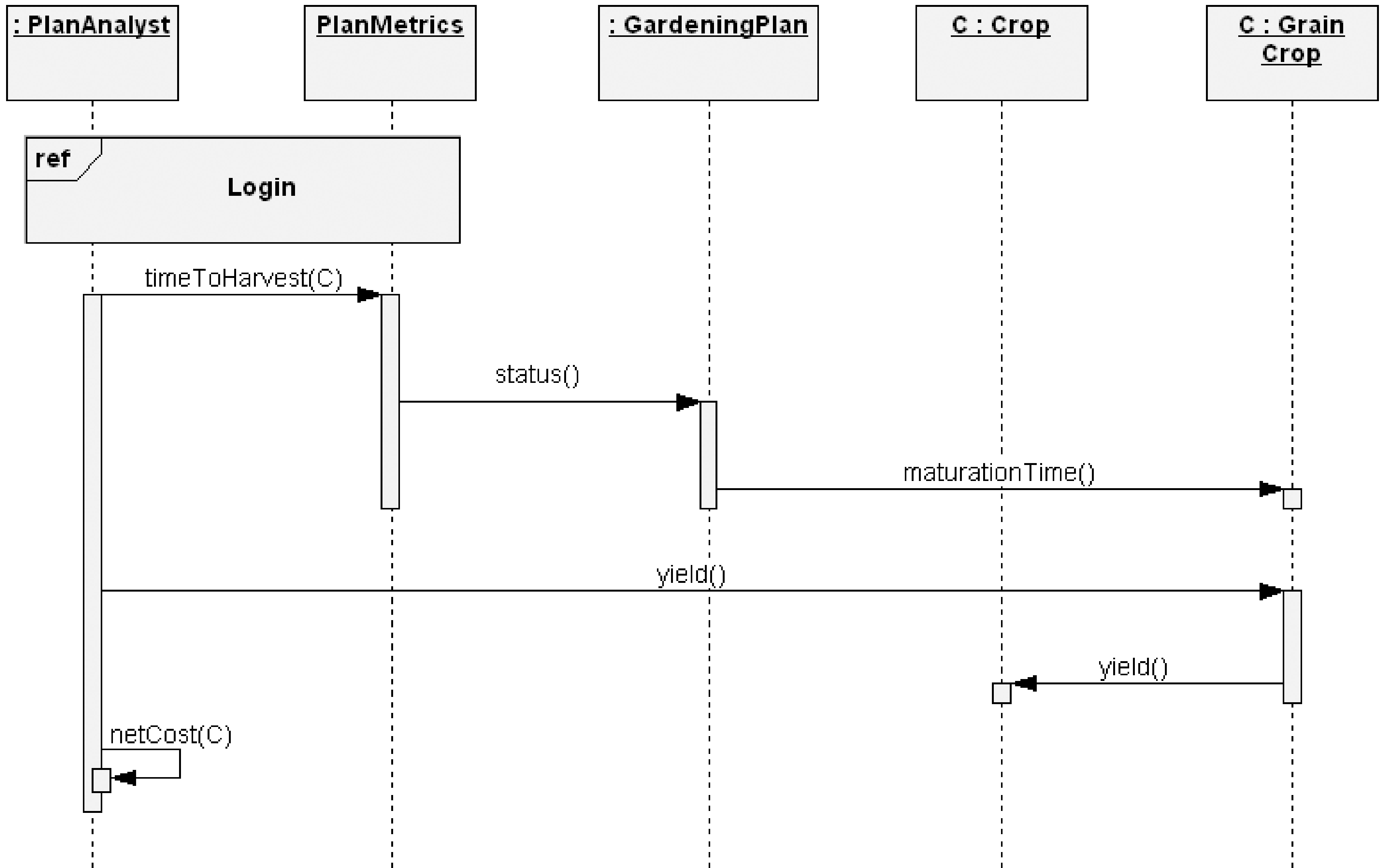
- We may adorn the vertical lines descending from each object in a sequence diagram with a box representing the relative time that the flow of control is focused in that object.



a sequence diagram for an emotion based music player

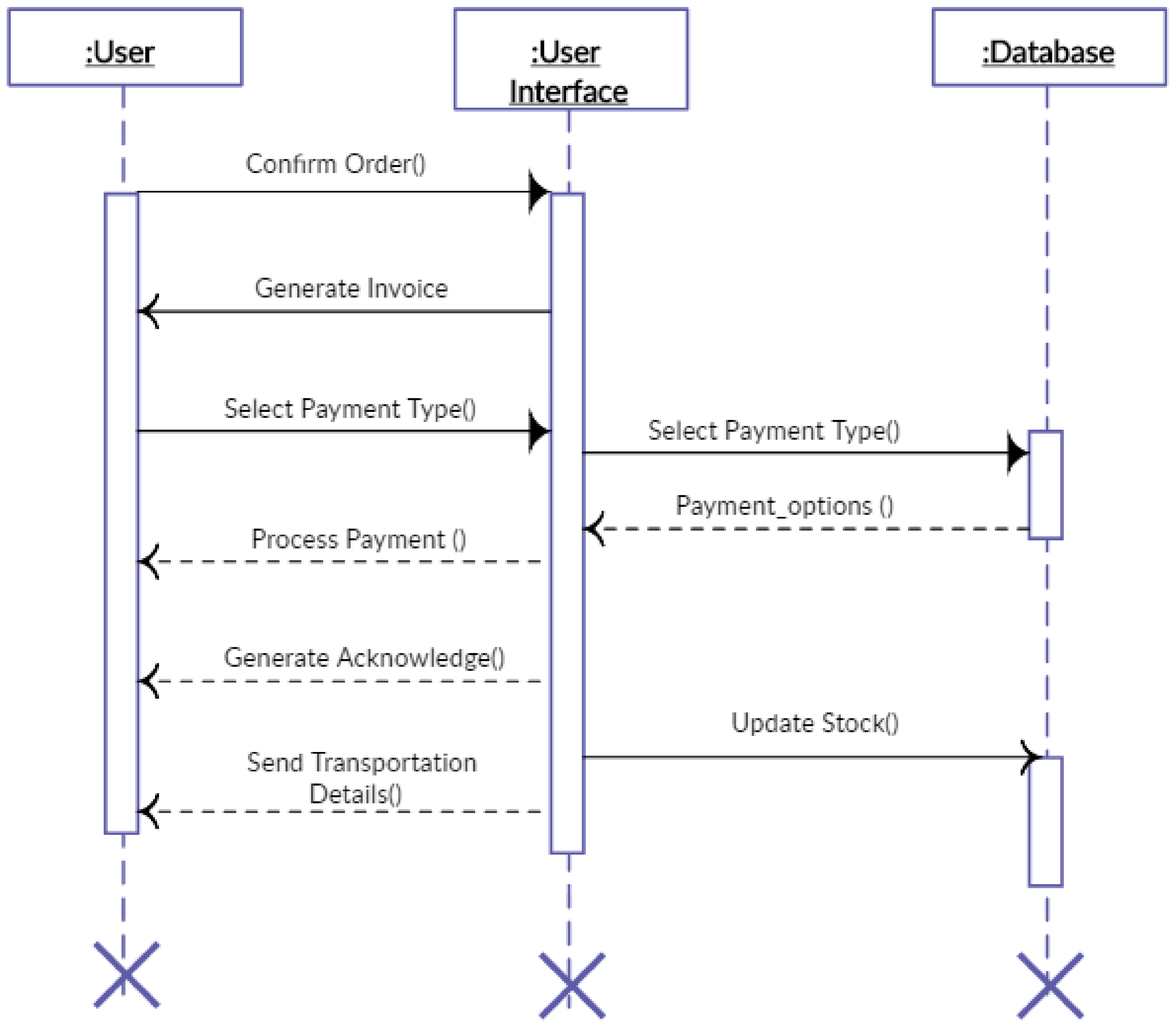
INTERACTION USE

- An interaction use is merely a way to indicate on a sequence diagram that we want to reuse an interaction that is defined elsewhere.

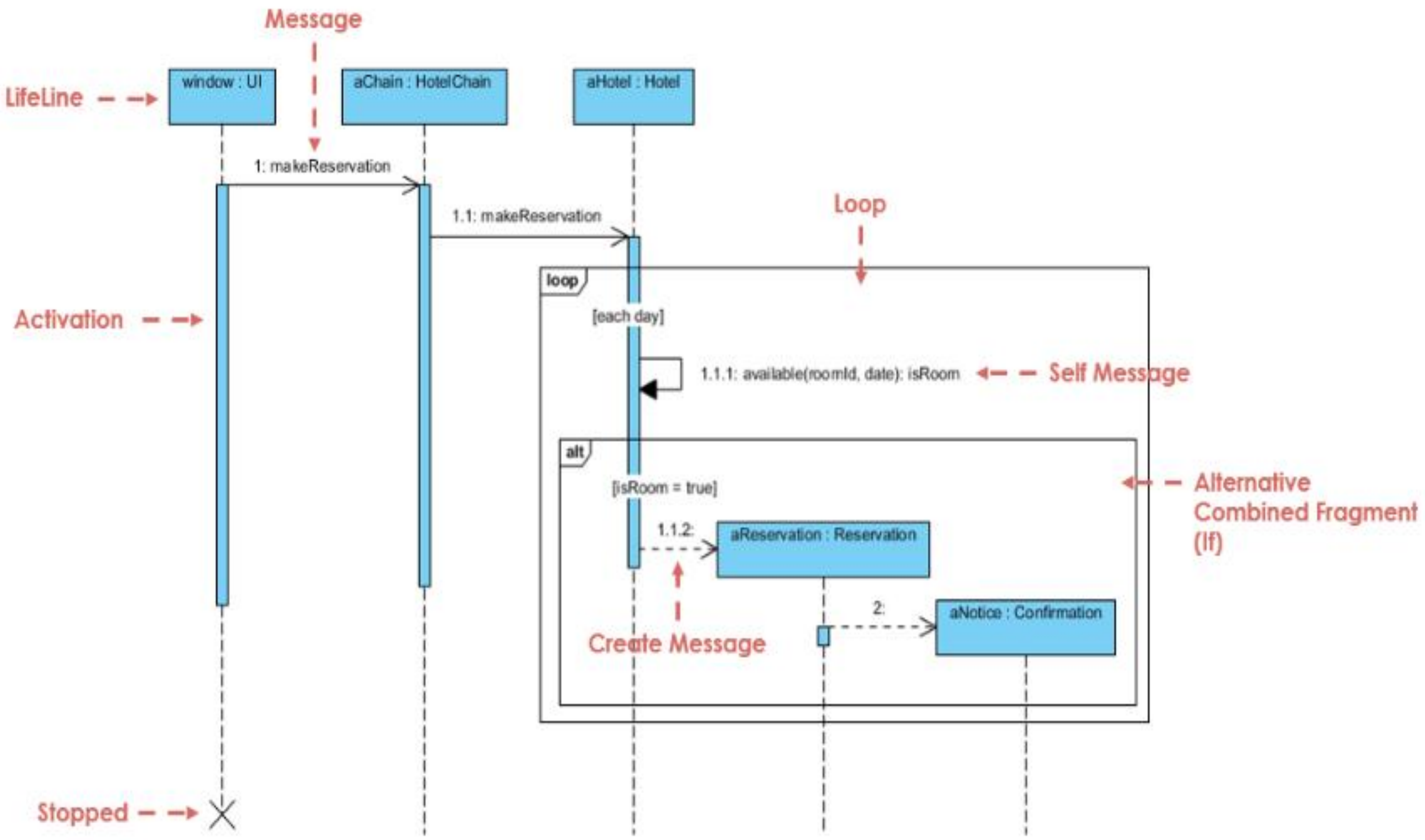


- In the above figure the frame labeled ref is used to show that a login Sequence is required before the PlanAnalyst uses the system.
- Or the frame, labeled ref, indicates that the Login sequence is inserted (i.e., copied) where this fragment is placed in this sequence.
- The actual login sequence would be defined on another sequence diagram.

Sequence Diagram for Order Processing



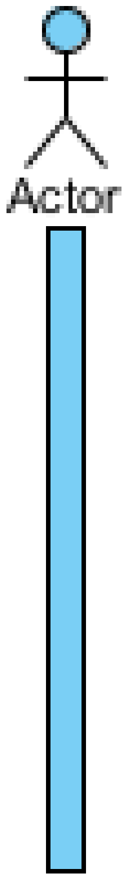
SEQUENCE DIAGRAM EXAMPLE: HOTEL SYSTEM



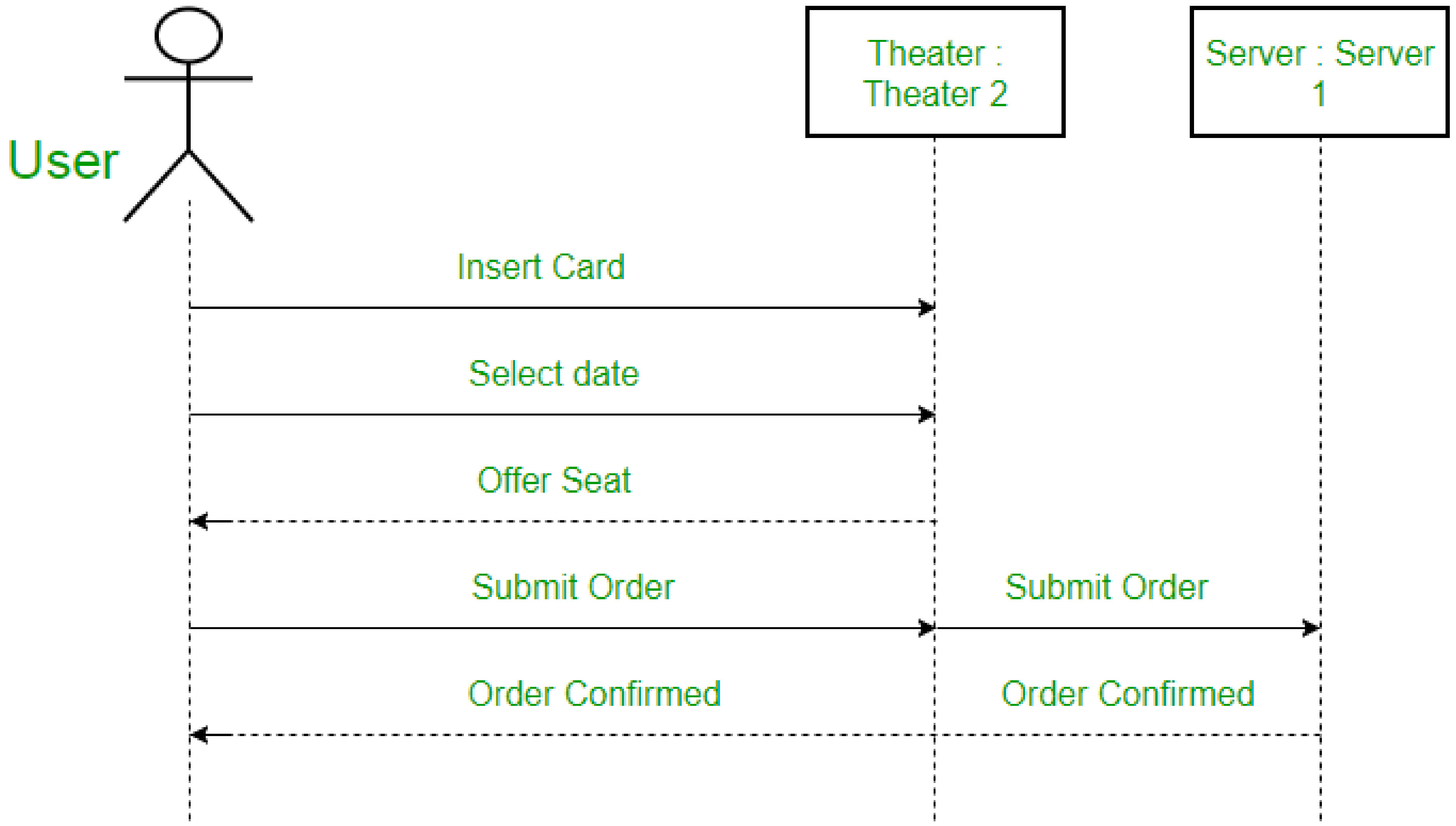
SUMMARY OF SEQUENCE DIAGRAM CONCEPT WITH THEIR NOTATION

❑ Actor

- a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data)
- represent roles played by human users, external hardware, or other subjects.

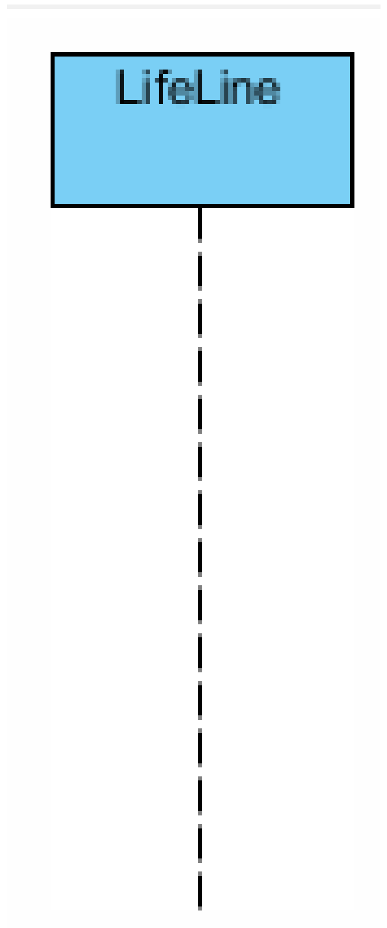


EXAMPLE USE OF ACTOR



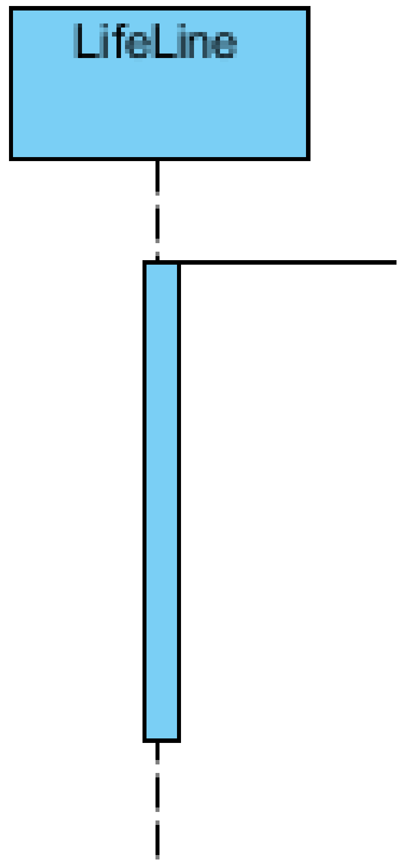
Lifeline

- A lifeline represents an individual participant in the Interaction.



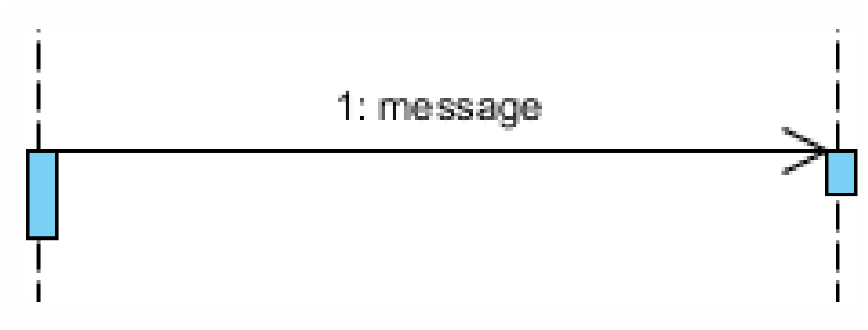
Activations

- A thin rectangle on a lifeline represents the period during which an element is performing an operation.
- The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively



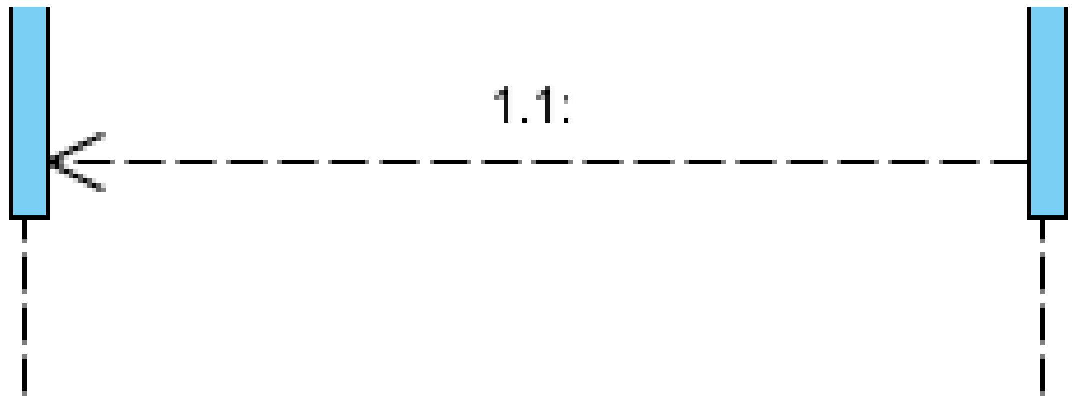
Call Message (Asynchronous)

- Call message is a kind of message that represents an invocation of operation of target lifeline.



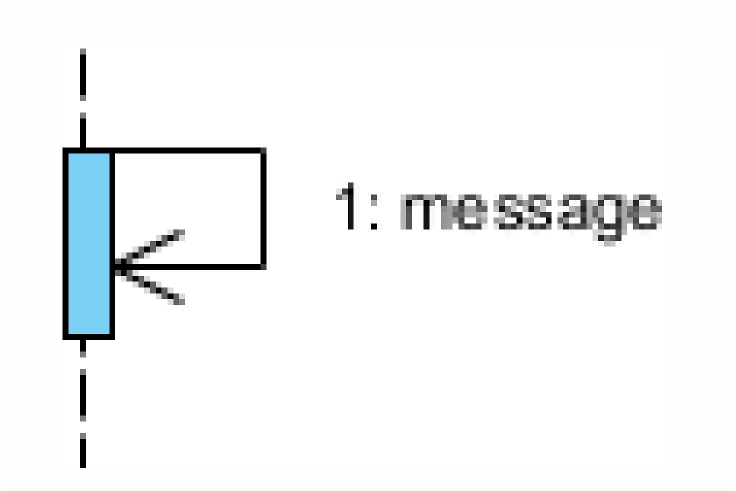
❑Return Message (For synchronous message)

- ❑Return message is a kind of message that represents the pass of information back to the caller of a corresponded former message.



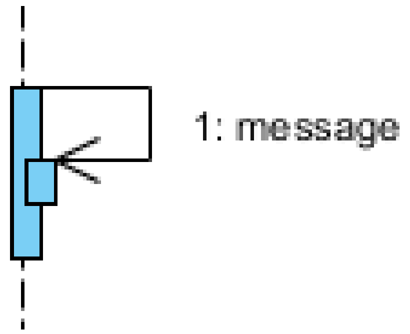
Self Message

- Self message is a kind of message that represents the invocation of message of the same lifeline.



Recursive Message

- Recursive message is a kind of message that represents the invocation of message of the same lifeline.
- It's target points to an activation on top of the activation where the message was invoked from.

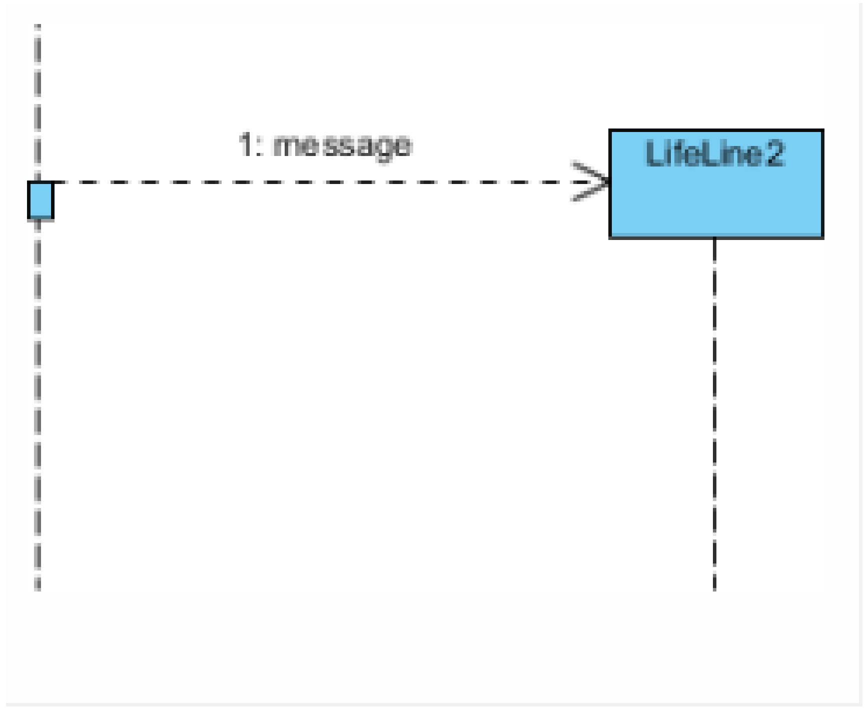


SELF MESSAGE VS RECURSIVE MESSAGE

- A message to *self* just means that the method that is to be invoked next happens to be in the same class of objects.
- A recursive call is a special case of a call to *self* in that it is to the same method (with a different state, so that it can eventually return out of the recursive calls).

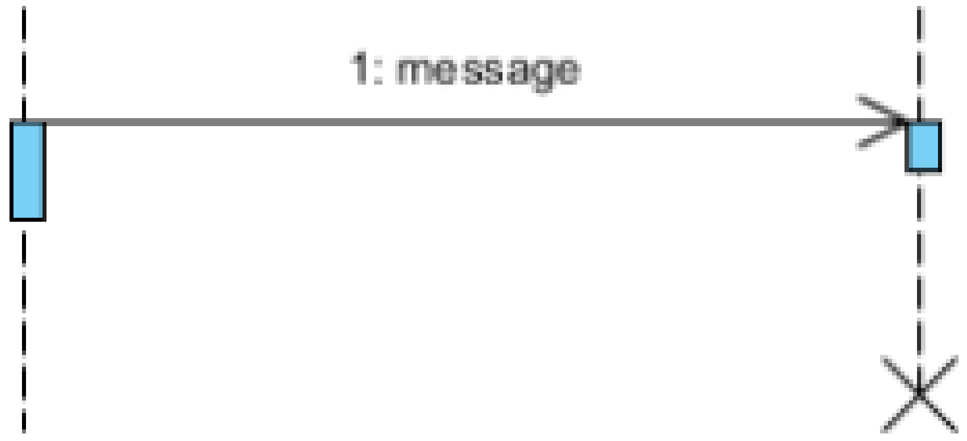
❑ Create Message

- Create message is a kind of message that represents the instantiation of (target) lifeline. Or
- We use a Create message to instantiate a new object in the sequence diagram



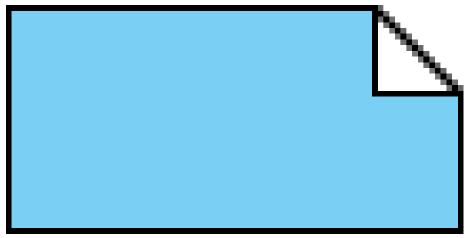
Destroy Message

- Destroy message is a kind of message that represents the request of destroying the lifecycle of target lifeline.



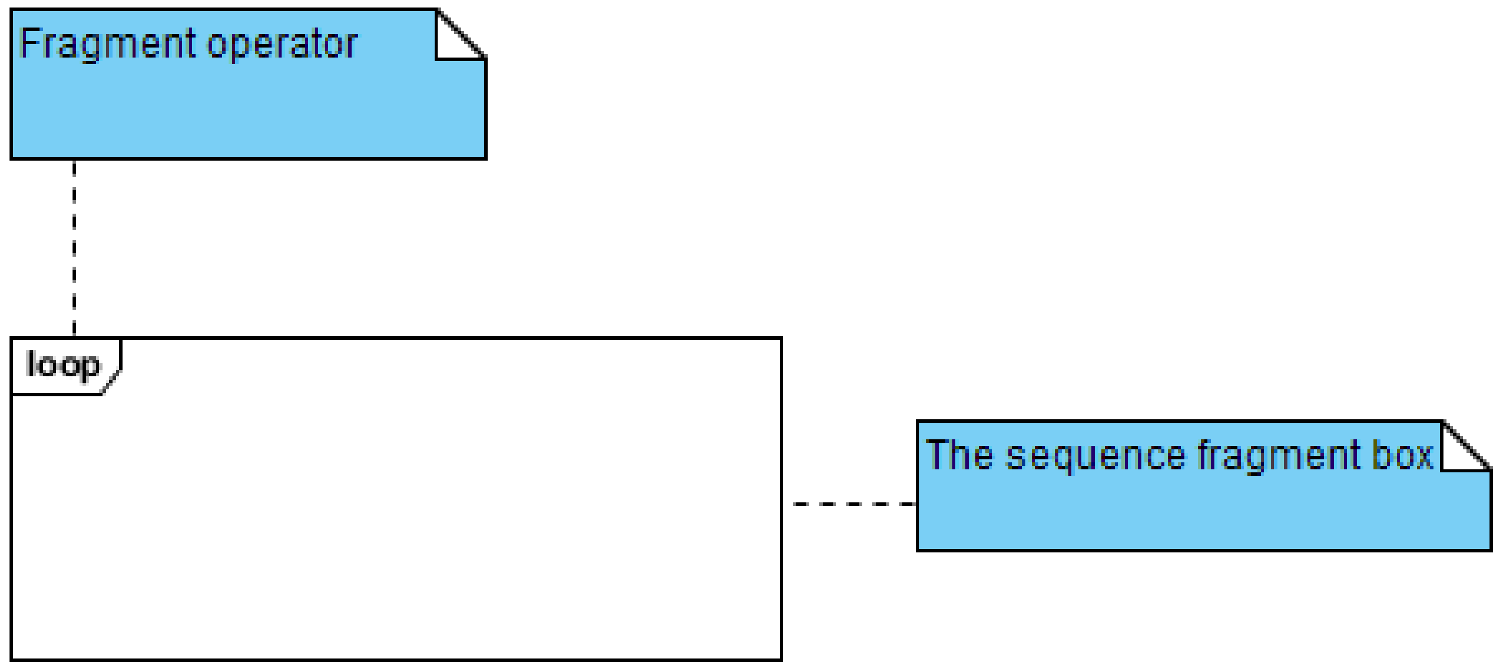
Note

- A note (comment) gives the ability to attach various remarks to elements.
- A comment carries no semantic force, but may contain information that is useful to a modeler.



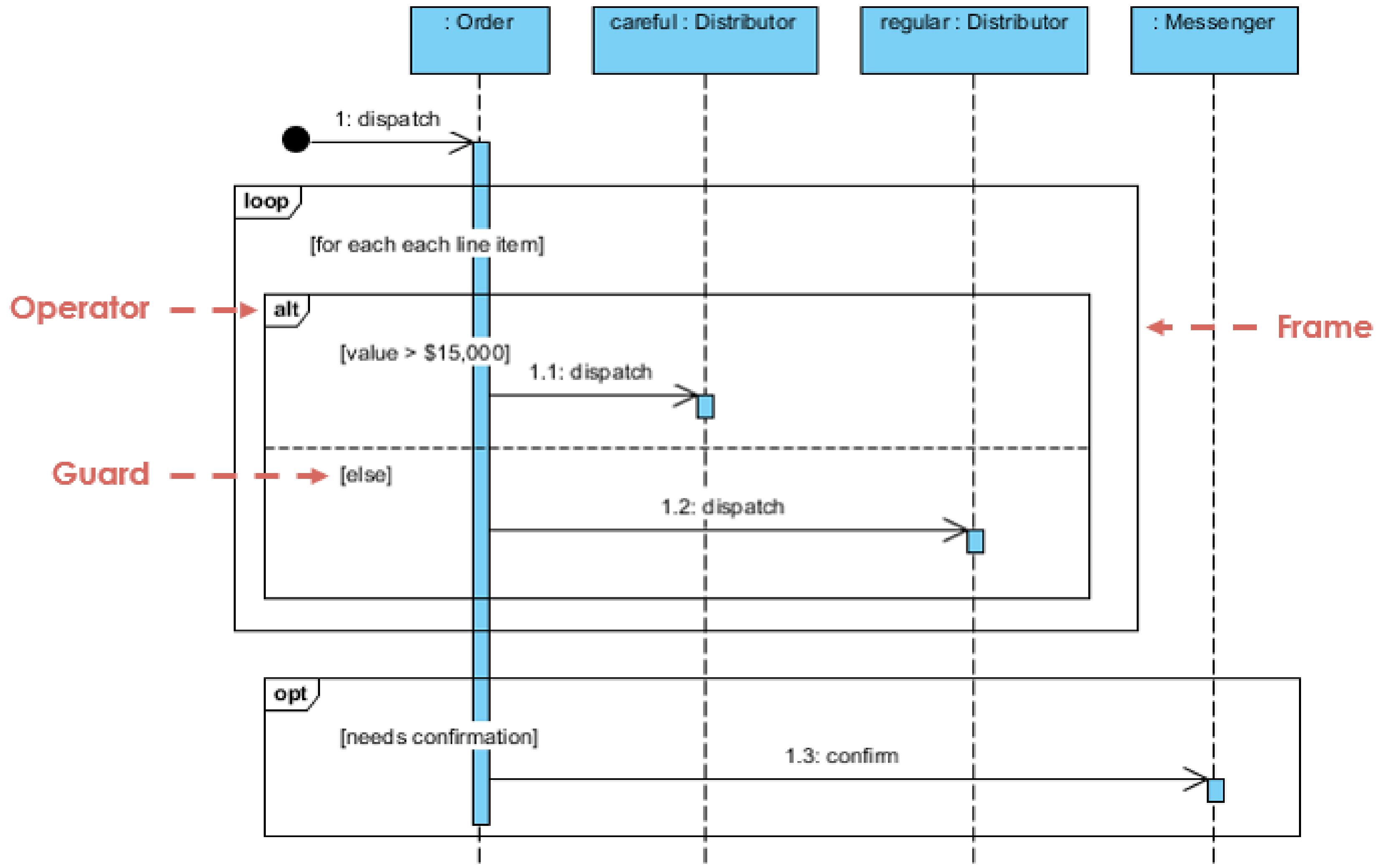
Sequence Fragments

- A sequence fragment is represented as a box, called a combined fragment, which encloses a portion of the interactions within a sequence diagram
- The fragment operator (in the top left cornet) indicates the type of fragment
- Fragment types: ref, assert, loop, break, alt, opt, neg



Operator	Fragment Type
alt	Alternative multiple fragments: only the one whose condition is true will execute.
opt	Optional: the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	Parallel: each fragment is run in parallel.
loop	Loop: the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	Critical region: the fragment can have only one thread executing it at once.
neg	Negative: the fragment shows an invalid interaction.
ref	Reference: refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	Sequence diagram: used to surround an entire sequence diagram.

- It is possible to combine frames in order to capture, e.g., loops or branches.
- Combined Fragment Example


: Order
careful : Distributor
regular : Distributor
: Messenger
1: dispatch
loop
[for each each line item]
Operator
alt
Frame
[value > $15,000]
1.1: dispatch
Guard
[else]
1.2: dispatch
opt
[needs confirmation]
1.3: confirm