

UML 2.0

Interaction Diagrams

UML 2.0

- Class diagrams (+ OCL constraints)
- Package diagrams
- Component diagrams
- Deployment diagrams
- Use case diagrams
- State diagrams
- Activity diagrams
- Interaction diagrams

UML 2.0

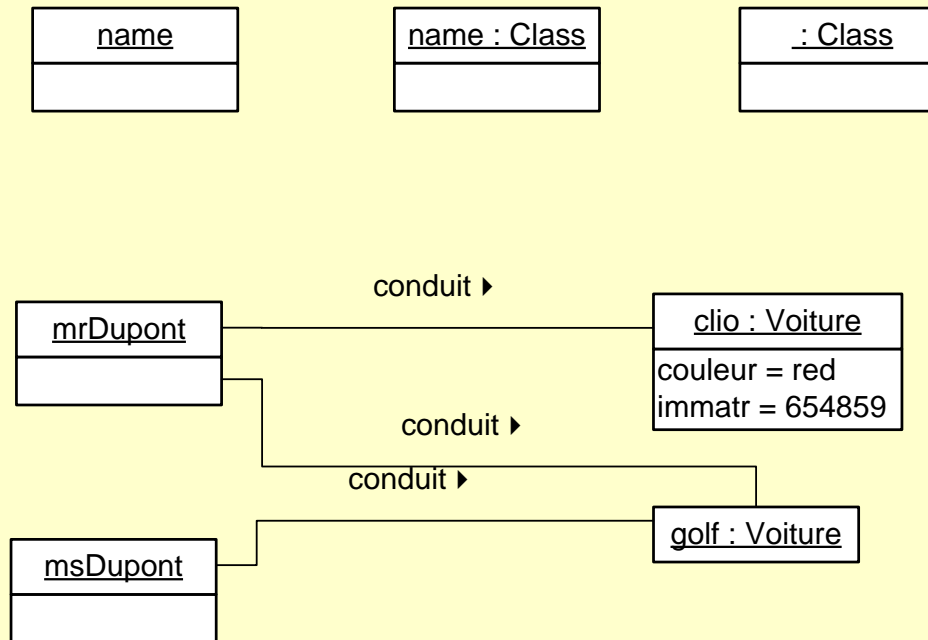
- ✓ Class diagrams (+ OCL constraints)
- Package diagrams
- Component diagrams
- Deployment diagrams
- ✓ Use case diagrams
- State diagrams
- Activity diagrams
- ✓ Interaction diagrams: today!

Recall: Class Diagrams

- Each UML diagram models a **different point of view** over the **same system**
- Example: Class Diagrams
 - **Goal**: to describe the static structure of the system, i.e. the system from a *spatial* point of view
 - **Specifications**: definition of classes, attributes, operations and relationships among classes
 - **Class**: definition domain for a set of objects which share the same characteristics
 - **Relationships**: aggregation, composition, generalization...

Recall: Object Diagrams

- Represent particular **instances** of class diagrams
- **Snapshot** of the system at a given point in time
- Graphical elements:



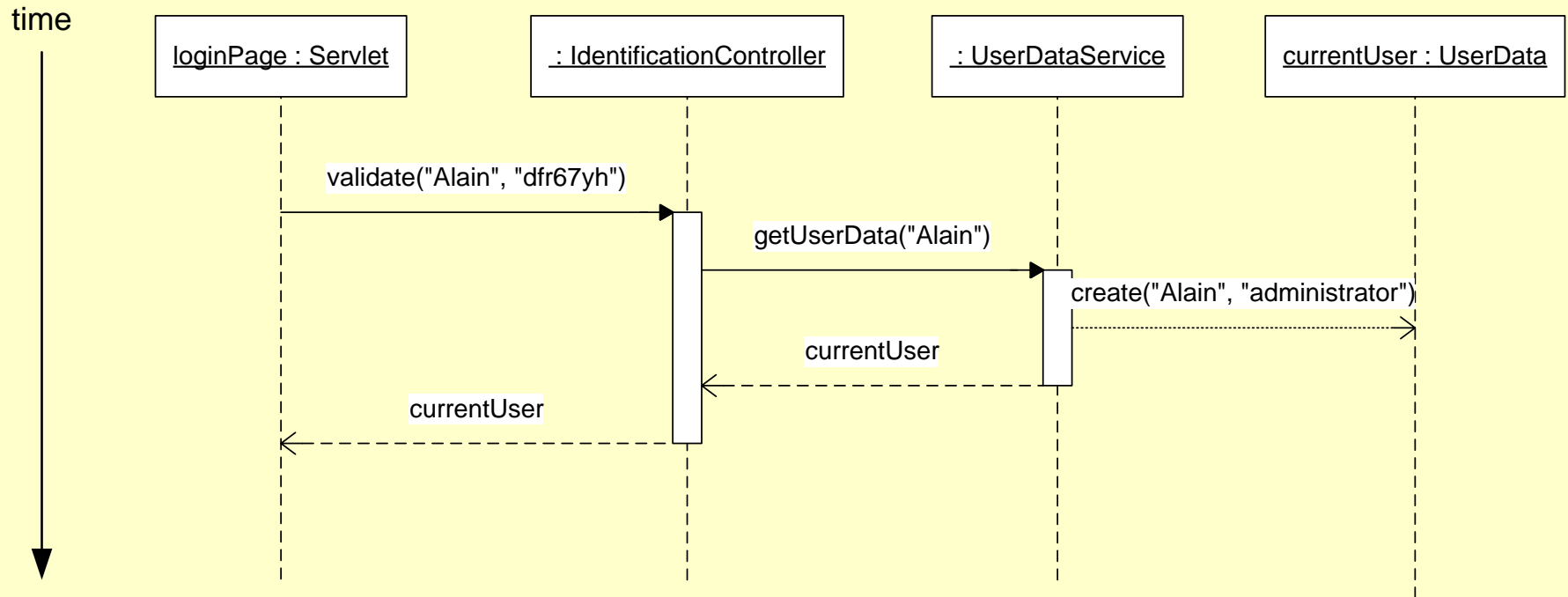
Recall: Use Case Diagrams

- Use Case Diagrams
 - **Goal**: describe the system from the users' point of view, i.e. identify the **needs**
 - **Specifications**: identifying actors, use cases, and the relationships among them
 - **Use case**: a feature of the system from the point of view of an external user
 - **Relationships**: generalize, include, extend

Interaction Diagrams

- Describe the **dynamic** structure of a system
- Represent **sequences of events** involving the objects of a system
- Model **communication protocols** between objects
 - Show interactions among the objects in a given situation
 - Sequence diagrams: interactions from “temporal” point of view (time matters)
 - Collaboration diagrams: interactions from “spacial” point of view (space matters)

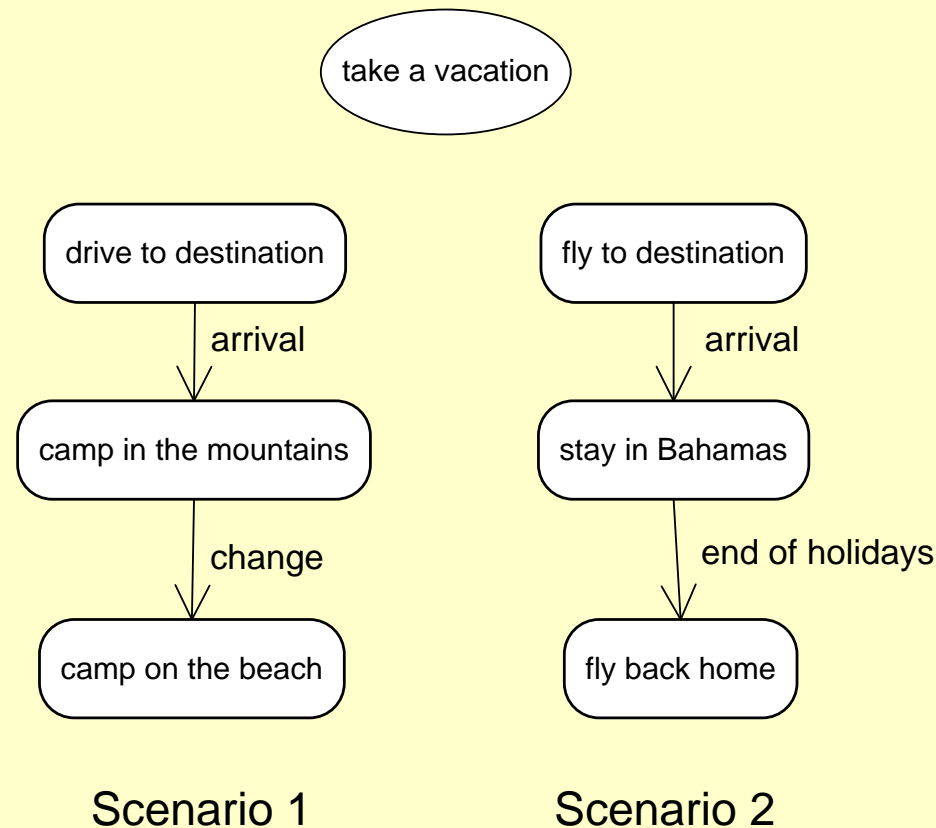
Sequence Diagram: an example



From Use Cases to Interaction Diagrams

- Use cases and Scenarios
 - A scenario is an instance of a use case
- Scenarios and Interaction Diagrams
 - Describe the interactions between the objects involved in the scenario

From Use Cases to Scenarios

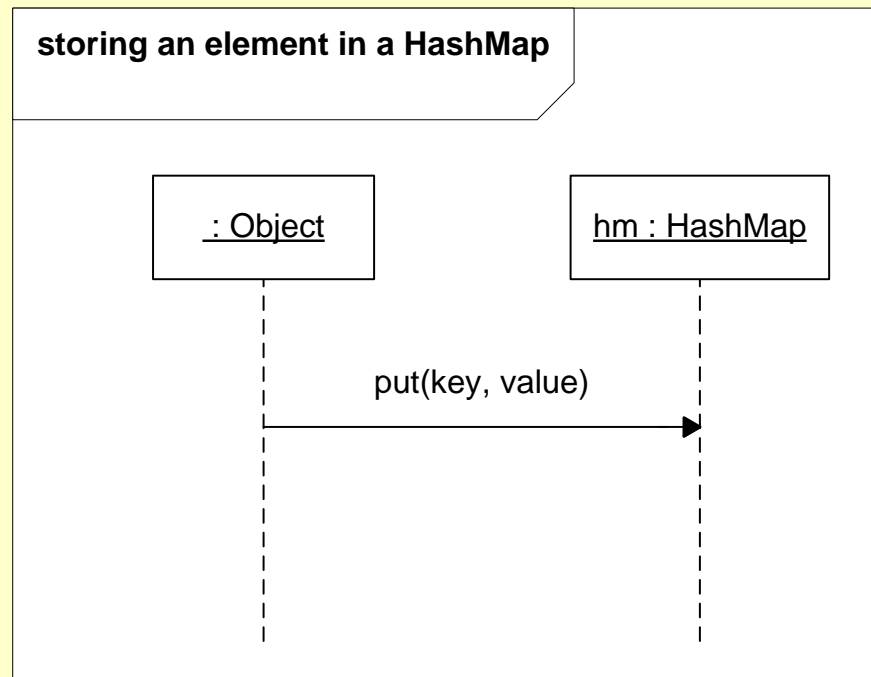


From Scenarios to Interaction Diagrams

- The interaction diagram shows how the objects involved in the scenario interact with each other
- An interaction describe the dynamic behavior of objects
- Two points of view are possible
 - Time (sequence diagram)
 - Space (collaboration diagram)
- The only possible way for an object A to interact with object B is to **send a message**.
 - `Class A { B.sendMessage() }`

Actors of an Interaction

- The actors of an interaction are **instances of objects**
- They are represented by objects **lifelines**
- The class of the objects can be specified or not

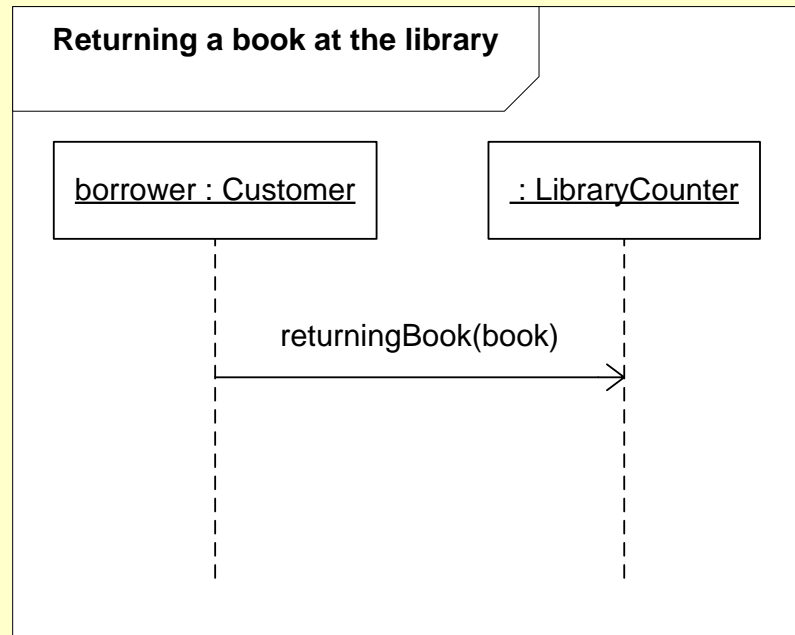


Messages

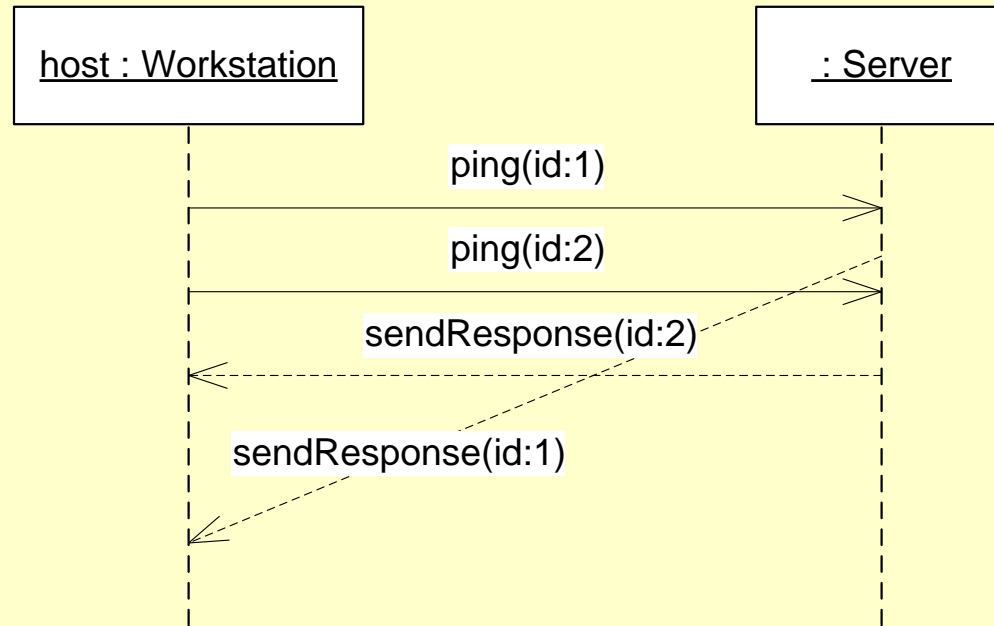
- Messages from an object to another are represented as arrows from the **sender** to the **receiver**
- Different shapes correspond to different types of messages:
 - Asynchronous: thick line with open arrow head
 - Synchronous (i.e. operation calls): filled arrow head
 - Reply message: dashed line
 - Object creation: dashed line with open arrow
 - Lost: a small circle at the arrow end of the line
 - Found: a small circle at the origin of the line

Asynchronous Messages(1)

- The sender **does not** stop and wait for the receiver to process the message and respond

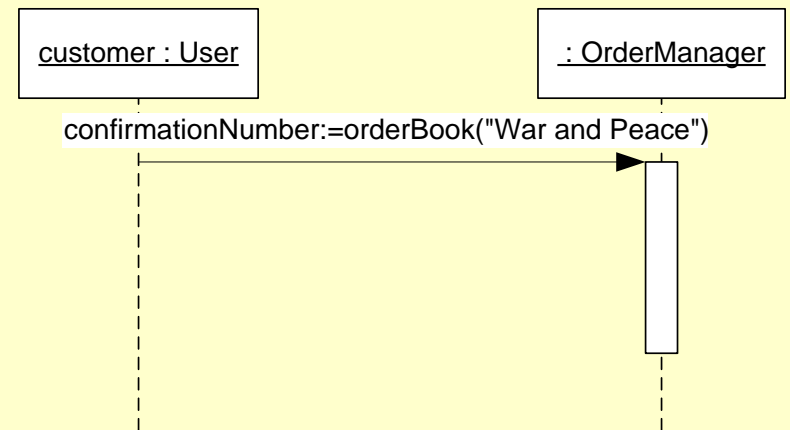
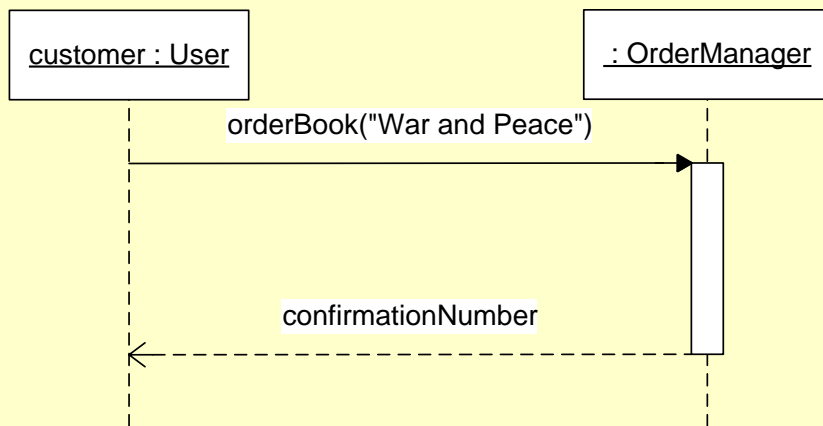


Asynchronous Messages(2)



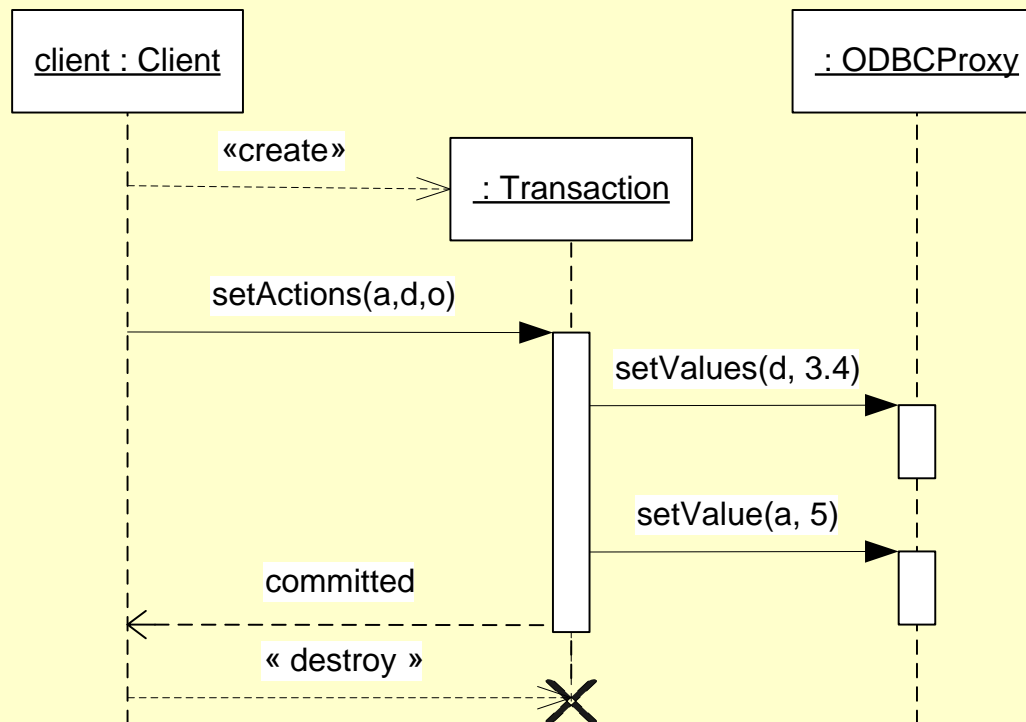
Synchronous Messages

- The sender stops and waits for the receiver to process the message and respond
- **Method calls** are (usually) synchronous messages
- The value returned is represented either by a dashed arrow, or specified as the return type of the message



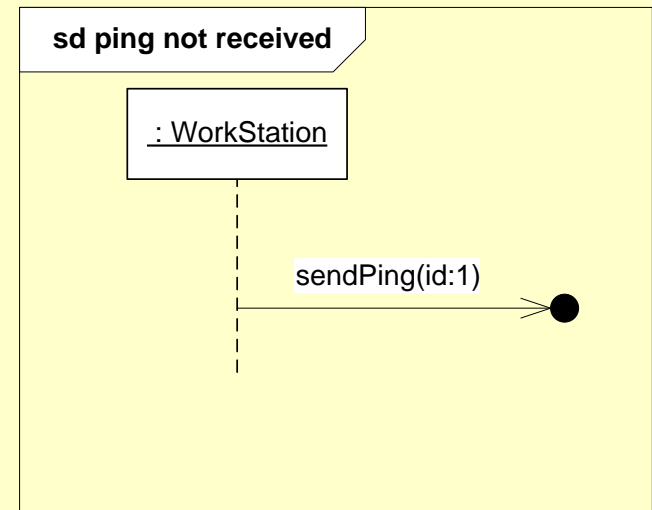
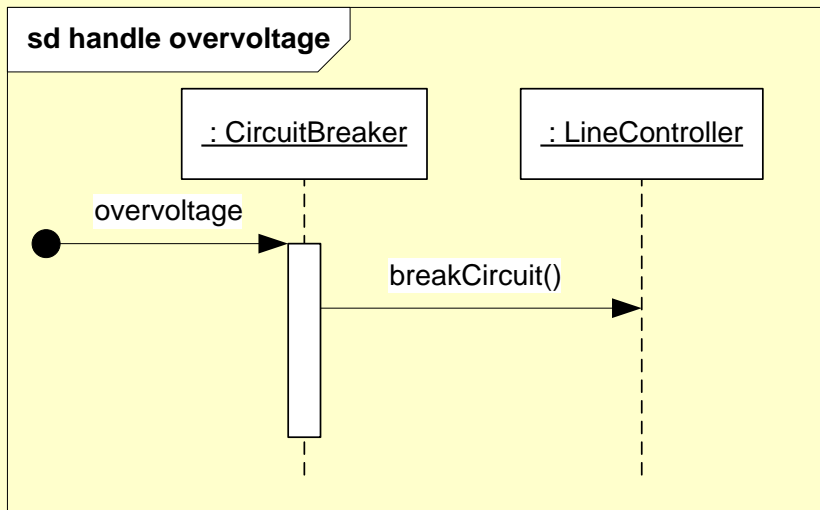
Destroy messages

- Messages provoking the dismissal of an object



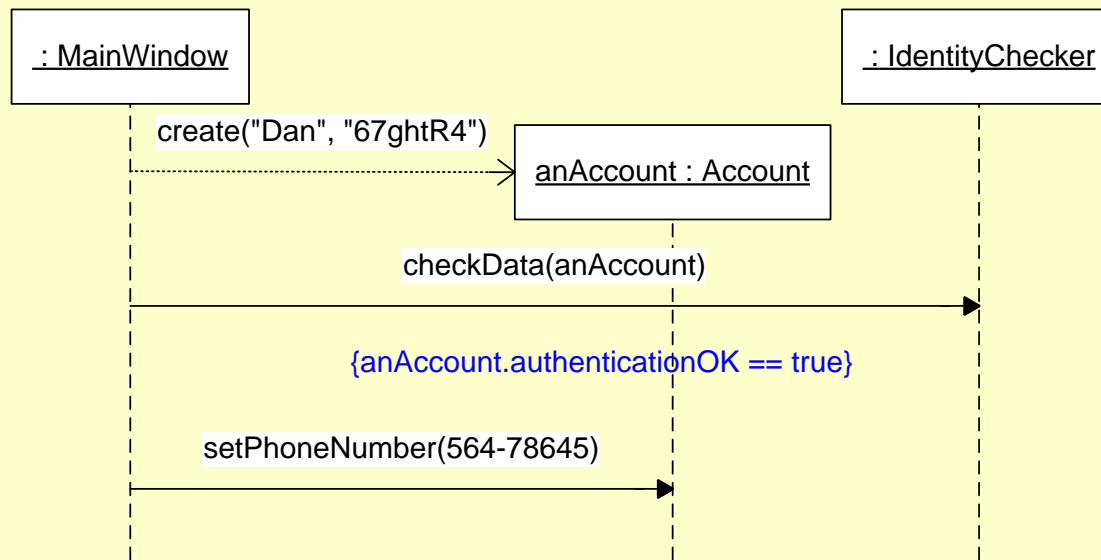
Lost & Found Messages

- Found message: the sender is unknown/unspecified
- Lost message: it never reached the receiver
- Lost/found are **relative** notions



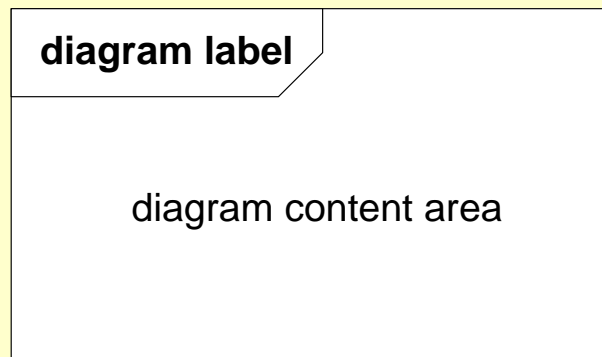
State Invariants

- State Invariants: conditions that must be true from a certain point onward



Frames

- Frames are named interaction fragments
- Special operators exist in order to specify special conditions for its execution (e.g. exclusive, alternative, conditional, parallel...)
- Introduced **for the first time** in UML 2.0 (they don't exist in UML 1.x)

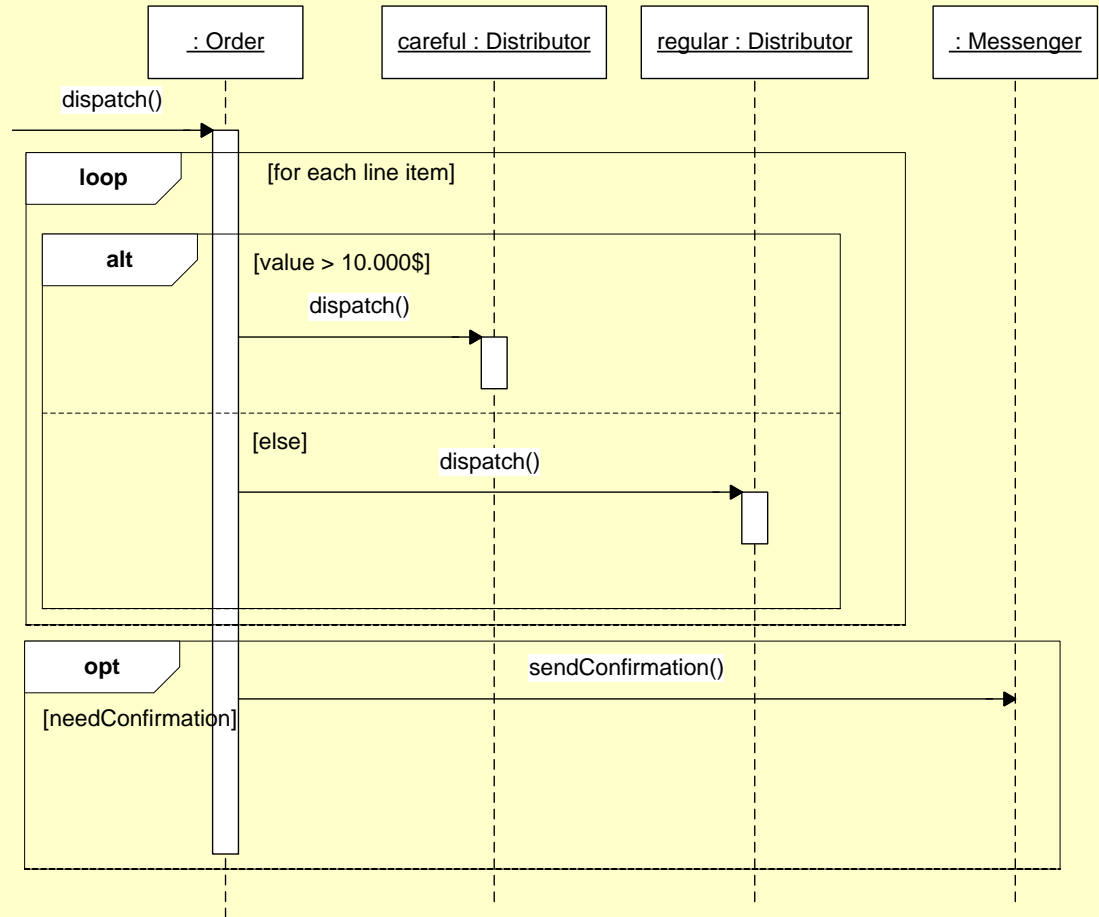


Frames operators

- **sd**: sequence diagram, used to surround and name an entire sequence diagram
- **alt**: alternative multiple fragments, only the fragment whose condition is true will execute
- **opt**: optional, if the condition is true the fragment will execute (equivalent to an Alt with only one trace)
- **par**: parallel, each fragment will run in parallel
- **loop**: the fragment may execute multiple times, controlled by a guard
- **critical**: 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 in another diagram

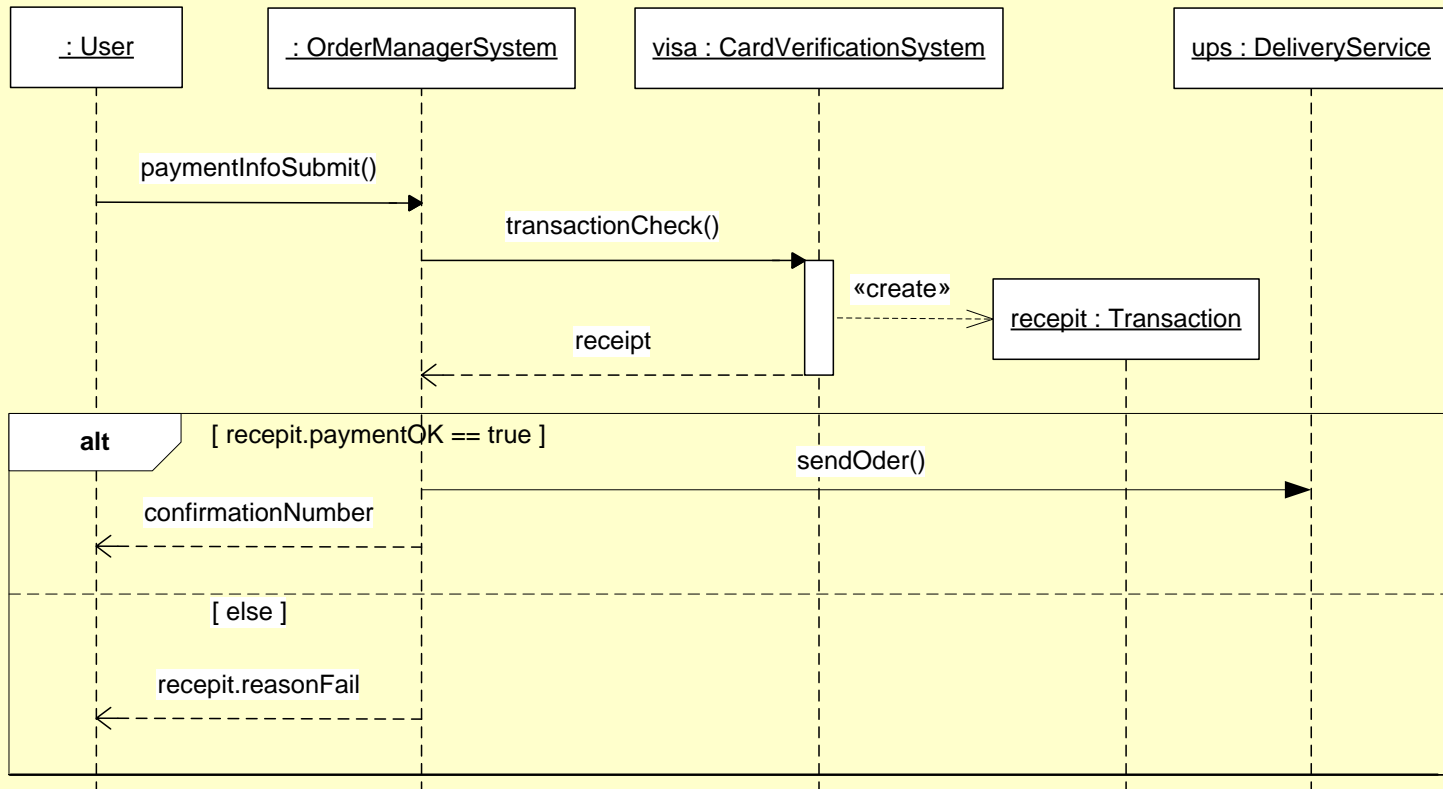
Frames: example

```
procedure dispatch
  foreach (lineitem)
    if (product.value>10k)
      careful.dispatch()
    else
      regular.dispatch()
    end if
  end for
  if (needsConfirmation)
    messenger.sendConfirm()
  end if
end procedure
```



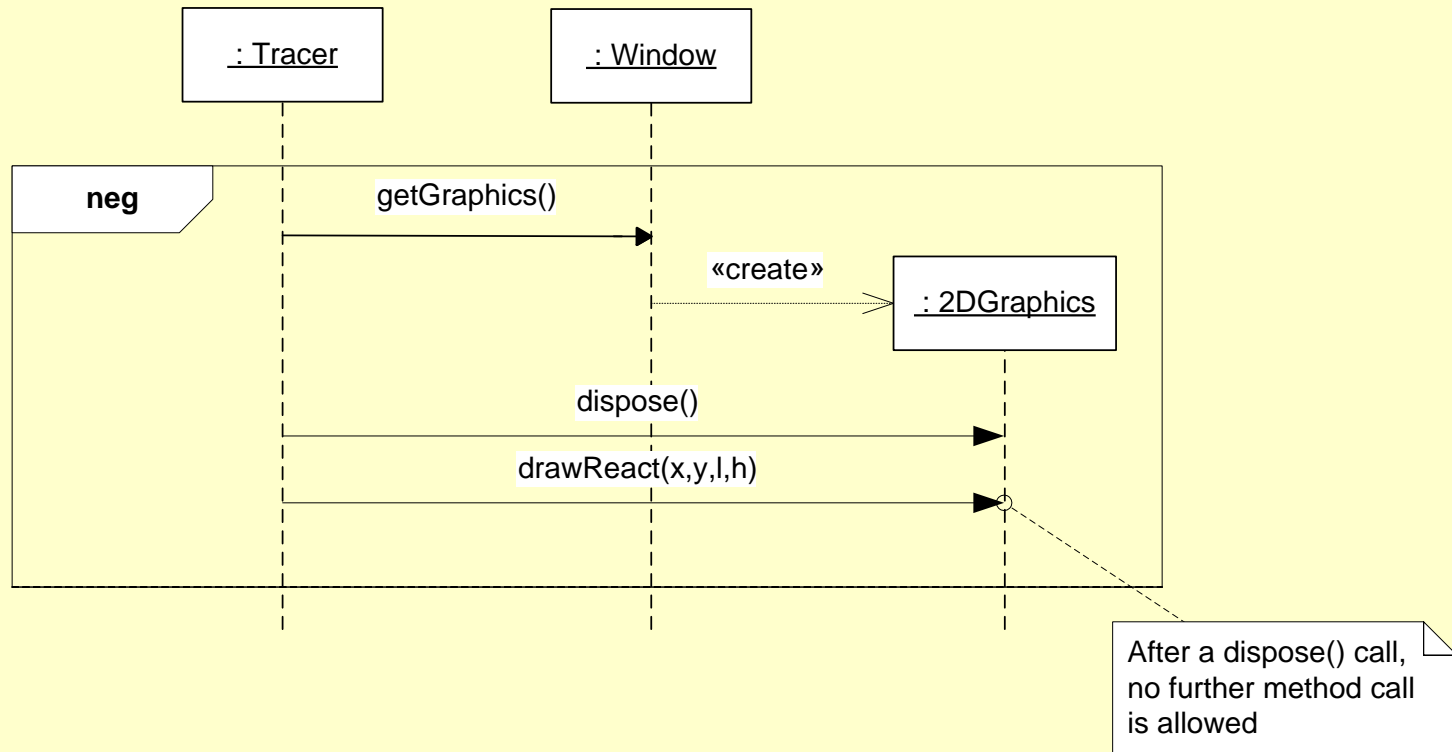
Frames: alt operator

- **alt**: selects one among several sequence fragments depending on a boolean guard



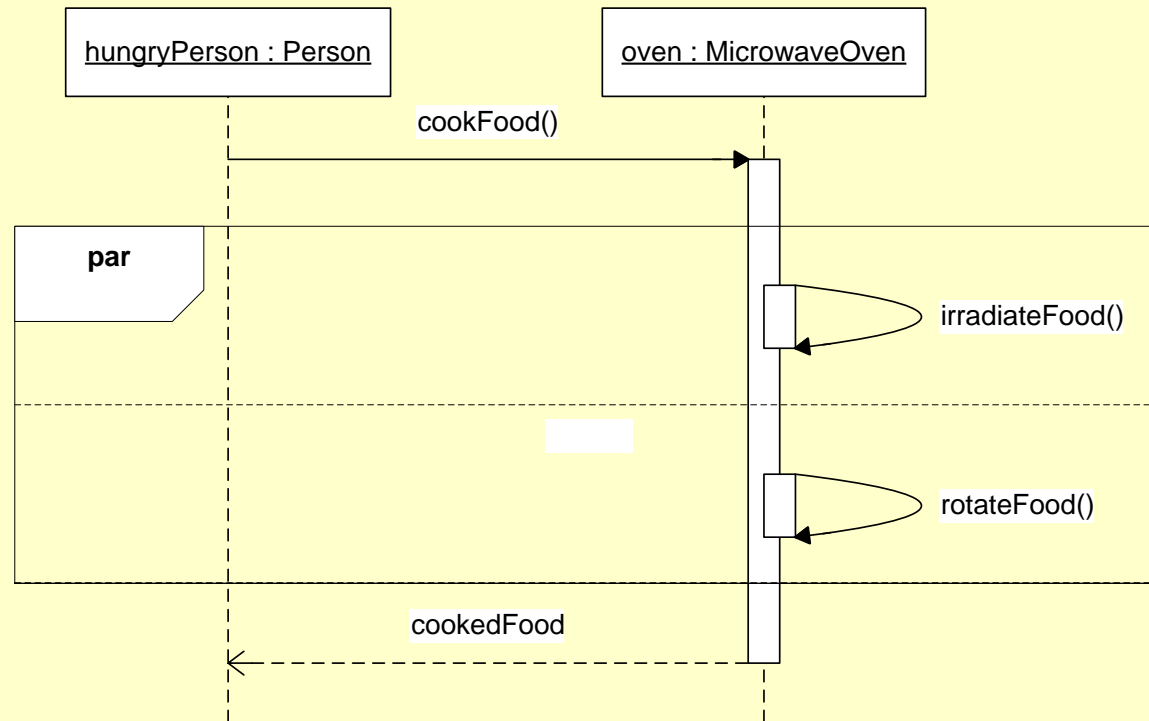
Frames: neg operator

- **neg**: describes an **invalid** sequence, usually associated with a comment explaining why it is forbidden



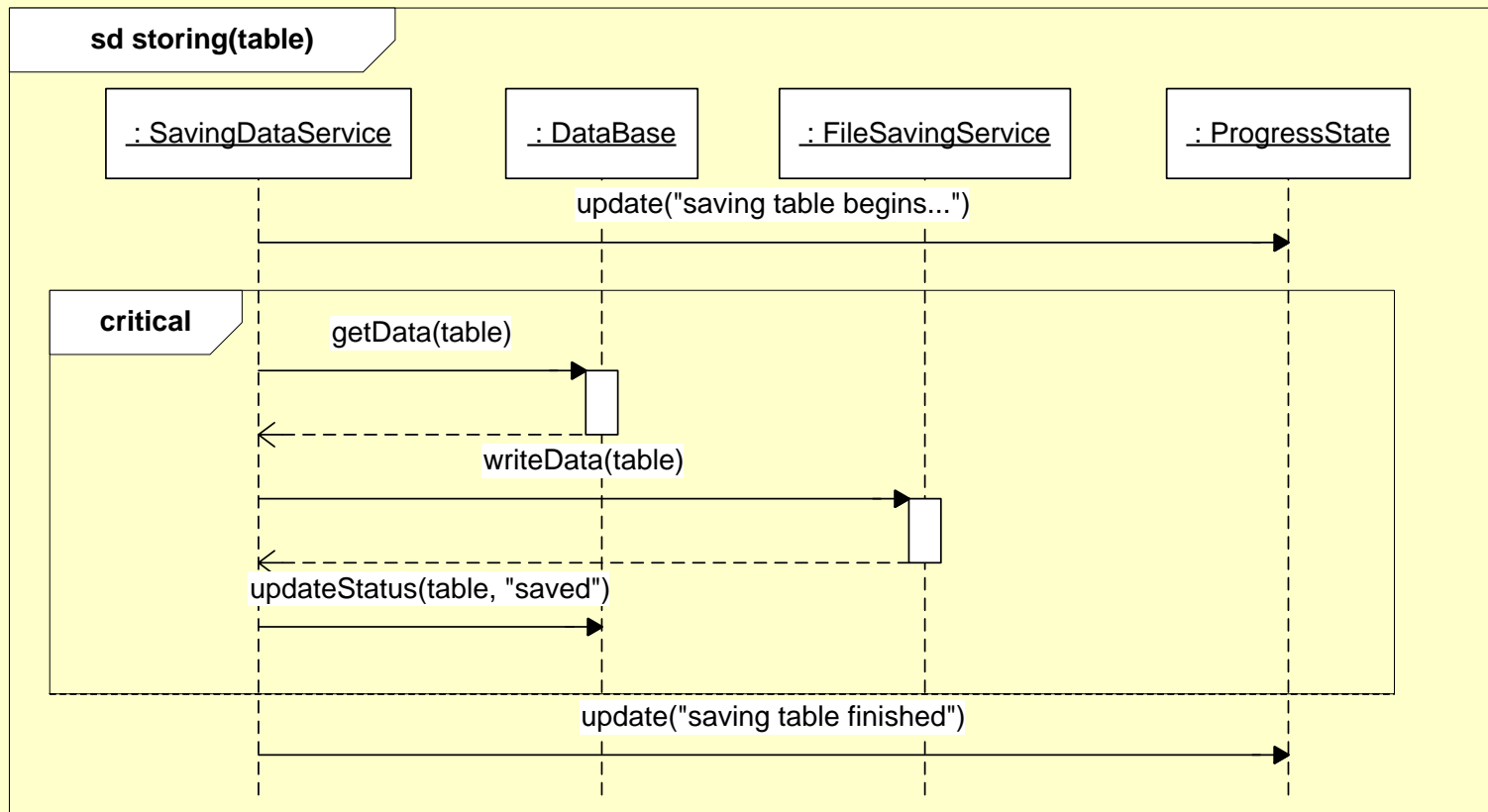
Frames: par operator

- **par**: the two sequences must be executed in parallel

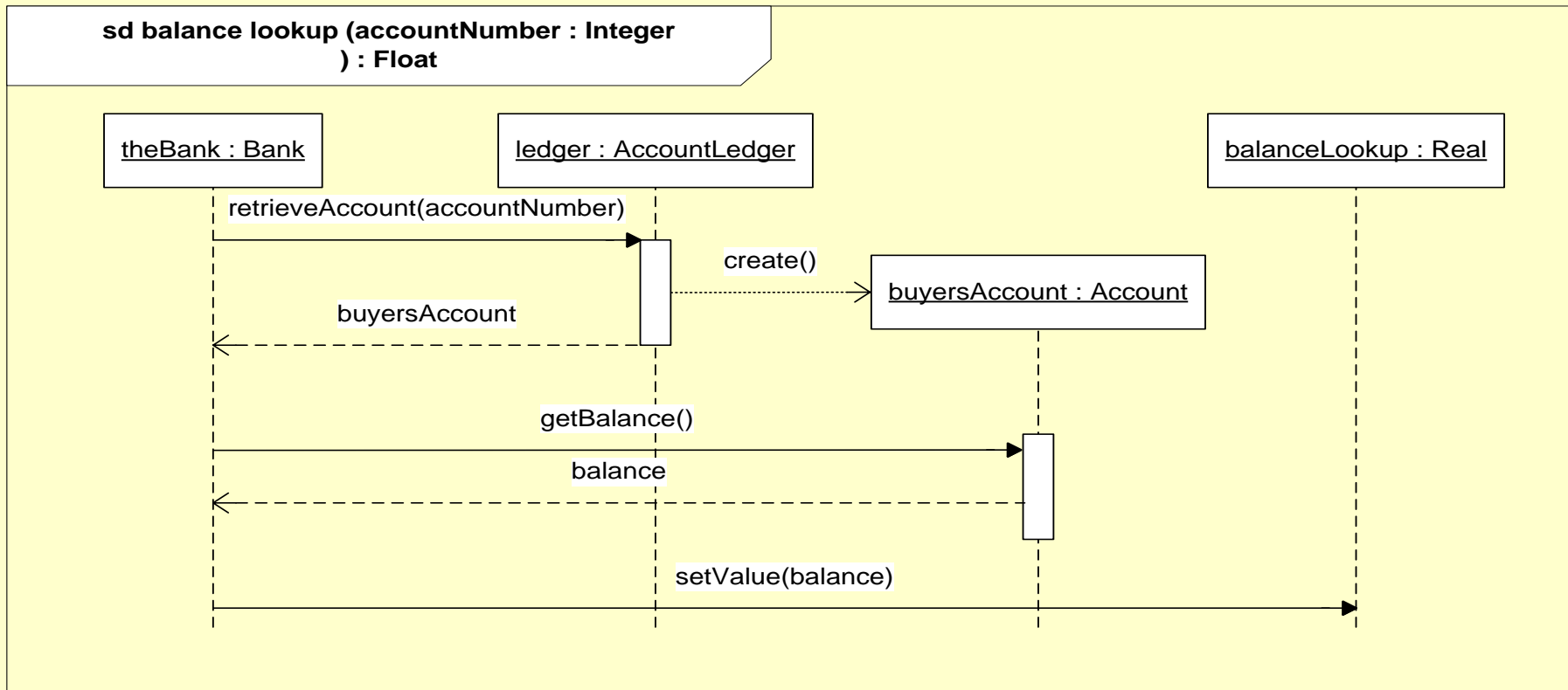


Frames: critical operator

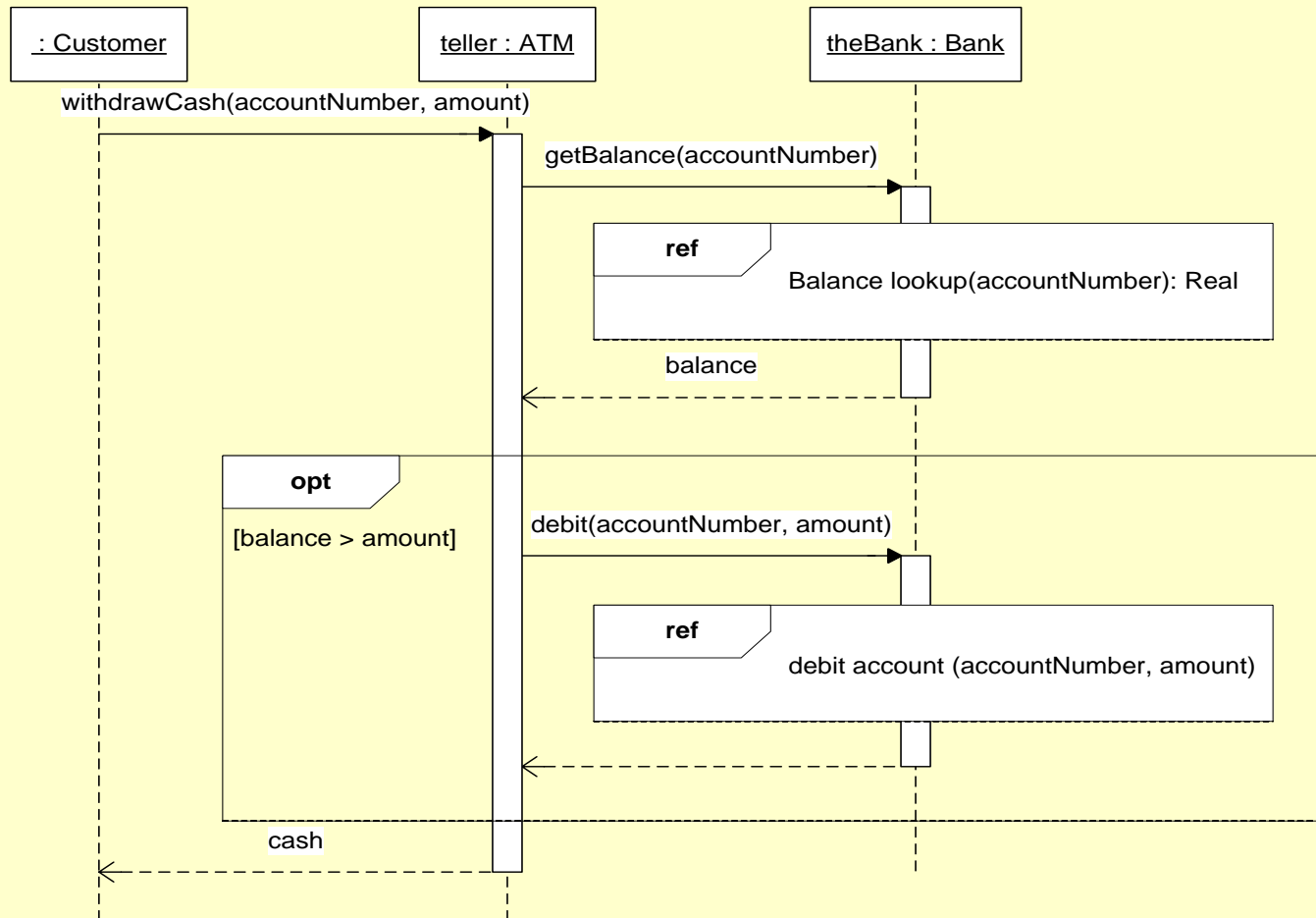
- **critical**: describes a sequence which must be executed as a single operation, i.e. atomically



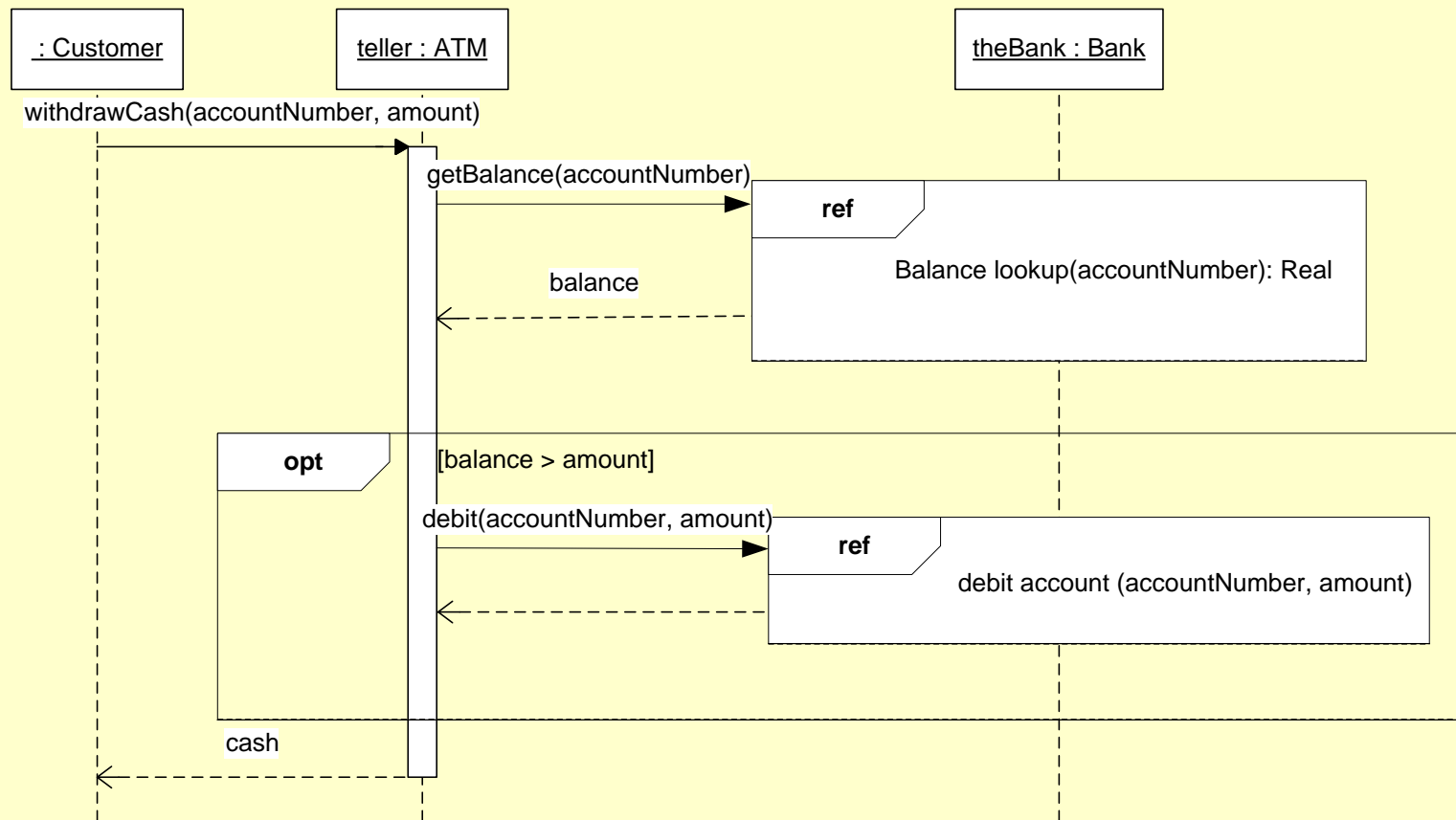
Frames: passing parameters



Frames: referencing other frames



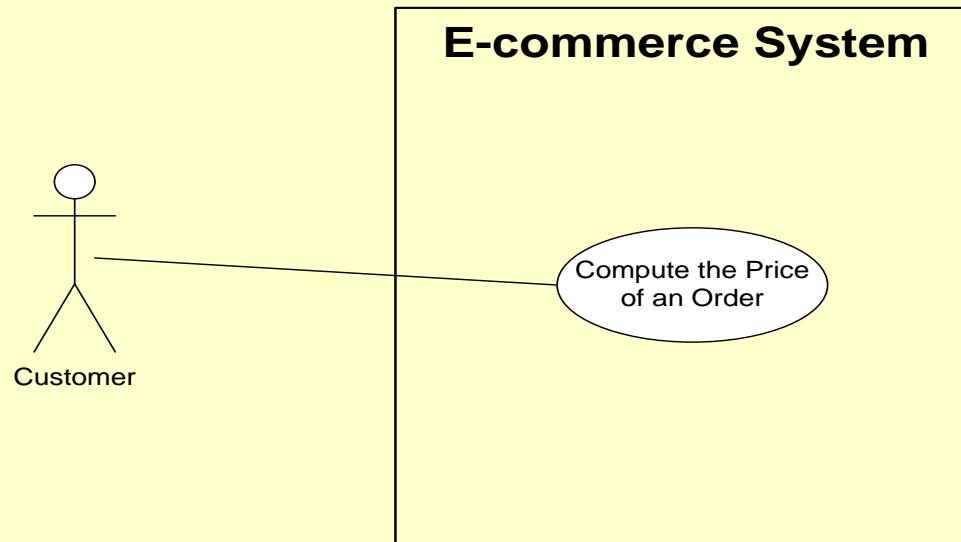
Frames: Gates



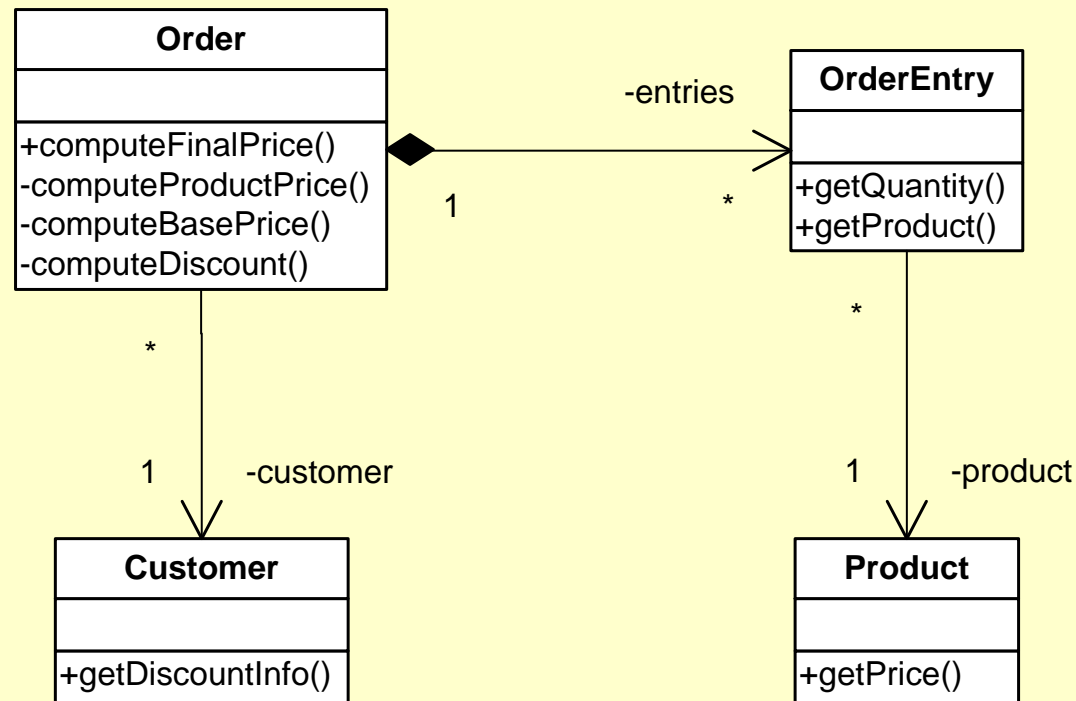
Example: Price of an Order

- A software application gives a customer the possibility to compute the final price of an order
- An order is made of a set of entries and is associated to only one customer
- An order entry is defined by
 - the product
 - the quantity of the product which is ordered
- A product stores some information about its price
- A customer can have special discounts

Compute the Price of an Order: Use Case



Compute the Price of an Order: Class Diagram



Compute the Price of an Order: Sequence Diagram

