Sequence Diagrams

Show Sequence of Interactions Between Objects

UML diagrams

There are many ways of organizing the UML diagrams.

Can be organized as the following:

1. **Structural diagrams:**
   - to show the building blocks of your
   - Ex: Class diagram
2. **Behavioral diagrams:**
   - to show how your system responds to requests or otherwise evolves over time.
   - Ex: Use case diagram
3. **Interaction diagrams:**
   - Is a type of behavioral diagram.
   - to depict the exchange of messages within a *collaboration* (a group of cooperating objects).
   - Ex: Sequence diagram & Collaboration diagram

Interaction Diagrams

- A series of diagrams describing the *dynamic behavior* of an object-oriented system.
  - A set of messages exchanged among a set of objects within a context to accomplish a purpose.
- Often used to model the way a use case is realized through a sequence of messages between objects.
- The purpose of Interaction diagrams is to:
  - Model interactions between objects
  - Assist in understanding how a system (a use case) actually works
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations and assign them to classes

Sequence Diagrams

- Illustrates how objects interacts with each other.
- Emphasizes time ordering of messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.

UML sequence diagrams

- ***Classifier*** abstract category of system modeling in UML (e.g. actor, object, interface, component, package, etc.)
- ***Messages*** are represented by an arrow. Describes a control flow in the system; it determines the sequence and place of execution of operations; messages are arranged according to the sequence of their appearance – the later they occur the lower they appear on the diagram
- ***Activations & Deactivation*** are represented by narrow rectangles. It shows time period during which the classifier performs an operation
- ***Lifelines*** are represented by dashed lines. The "X" mark at the end of the lifeline indicates the point at which the object ceases to exist in the system

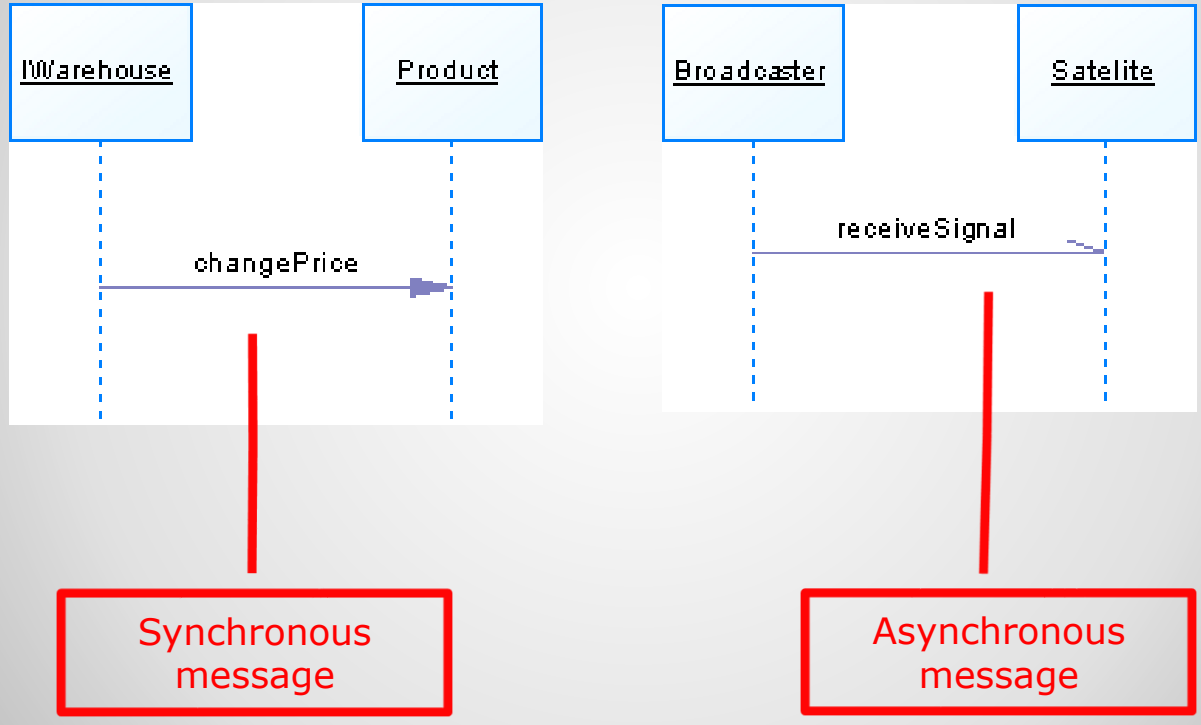
- Types of messages
- Creating and destroying objects

Sequence Diagrams

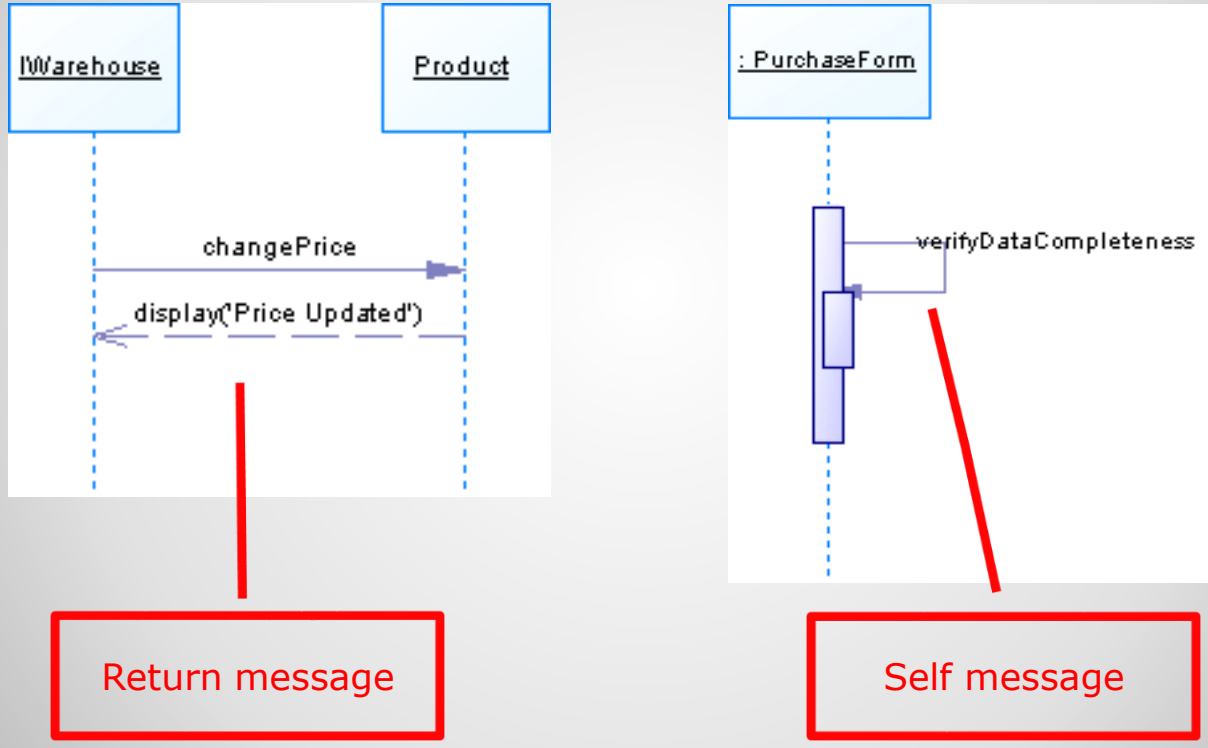
Message types:

- **synchronous message –** passes control from the sender classifier to the receiver classifier
- **asynchronous message –** does not pass control, does not wait for an answer from the receiver, may continue processing
- **return message –** indicates control return to the sender classifier after synchronous message and may also initiate a certain operation
- **self message** – message sent by the classifier to itself resulting in calling its own operation; self message is a certain kind of iteration, which creates a nested execution specification

Sequence Diagrams



Sequence Diagrams – Message



Sequence Diagrams – Message

Other message types:

- **lost message** – message sent from a known sender to an unknown receiver (temporary message)
- **found message** – message whose sender is unknown (may be an external signal, stimulus)
- **balking message** – message which will not be handled by the receiver classifier if it cannot be handled immediately
- **timeout message** – similar to balking message although sender classifier is willing to wait for handling the operation for a specified period of time

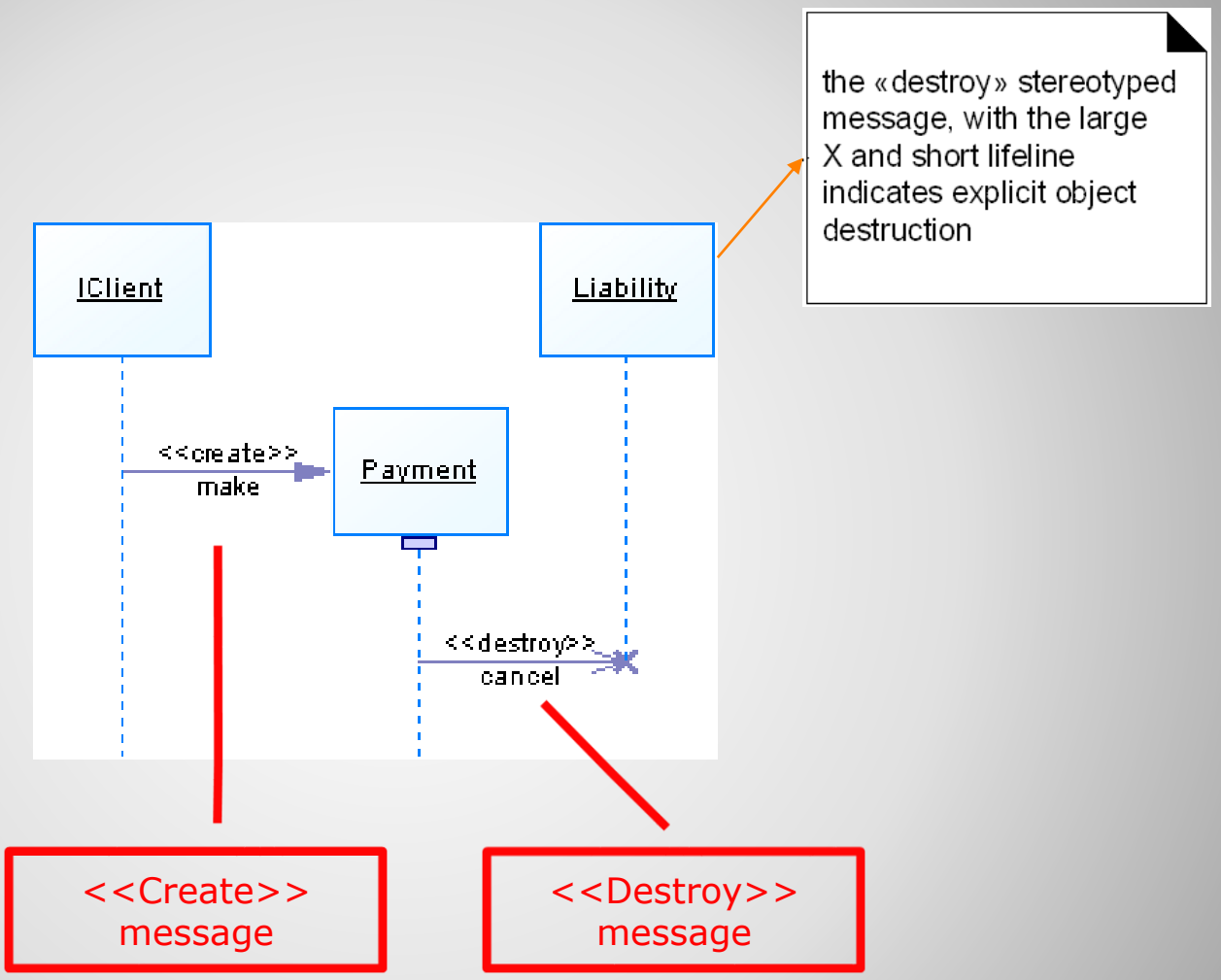
Sequence Diagrams – Message

Sequence Diagrams – Objects

Creating and destroying objects:

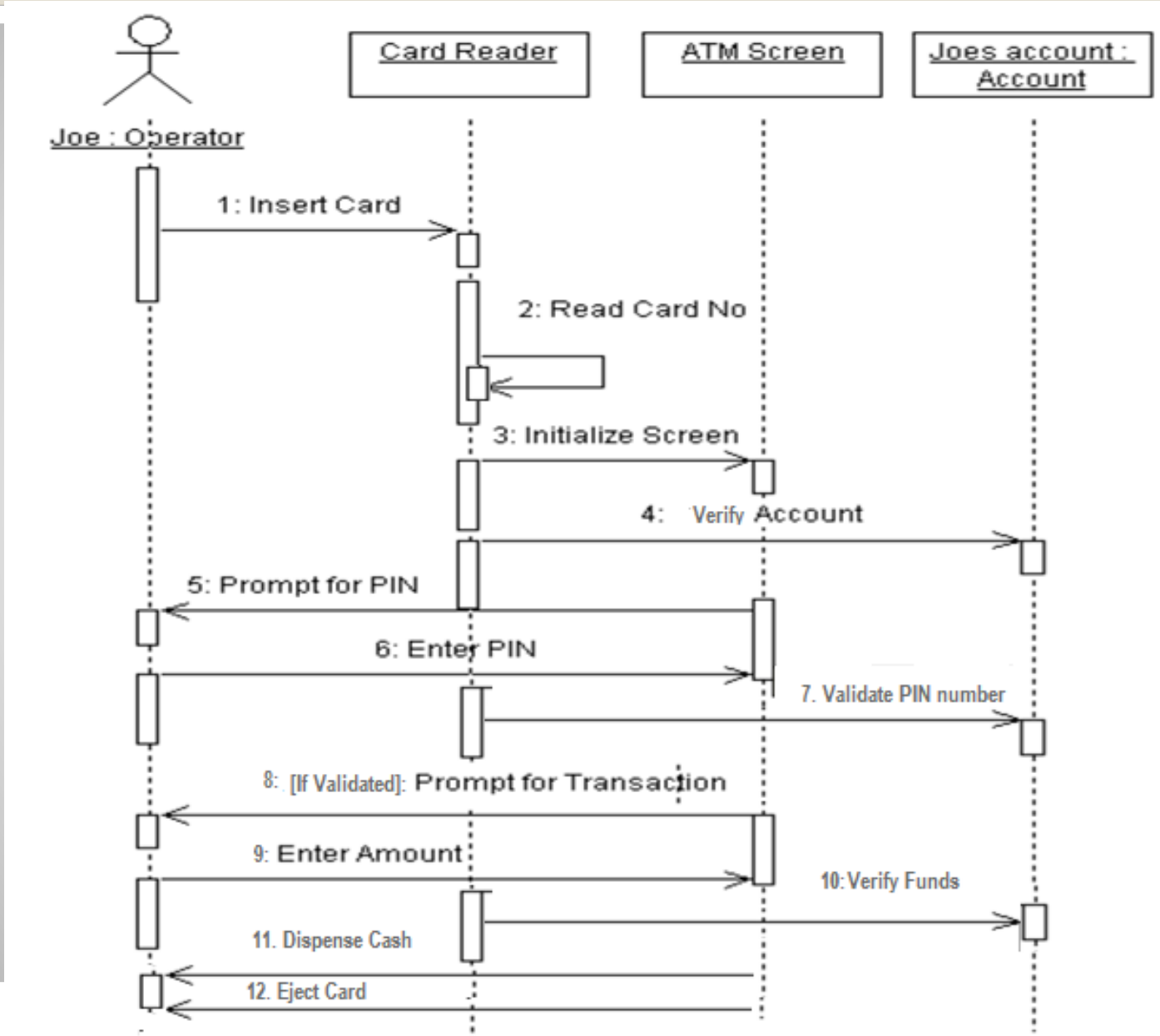
- **"create" stereotype message** – results in creation of an object, which is situated below the primary existing classifiers, corresponding with the time of its creation
- **"destroy" stereotype message** – results in destruction of an object

Sequence Diagrams – Objects



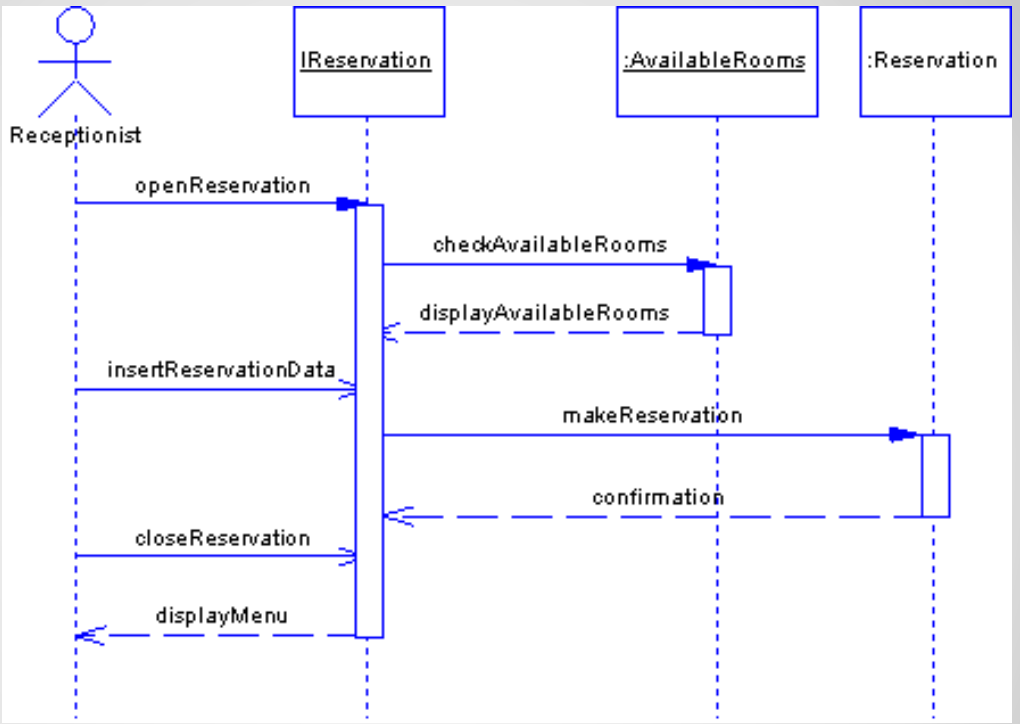
Example for Withdraw

- Joe withdraws $20 from the ATM (flow of events)
  - The process begins when Joe inserts his card into the card reader. The card reader reads the number on Joe's card, then tells the ATM screen to initialize itself
  - The card reader verifies the card against account and prompts Joe for his PIN.
  - Joe enters PIN.
  - Joe's PIN is validated and the ATM Screen prompts him for a transaction
  - Joe selects Withdraw Money
  - The ATM Screen prompts Joe for an amount.
  - Joe enters $ 20.
  - The card reader verifies that Joe's account has sufficient funds and subtracts $ 20 from his account.
  - The ATM dispenses $ 20 and ejects Joe's card

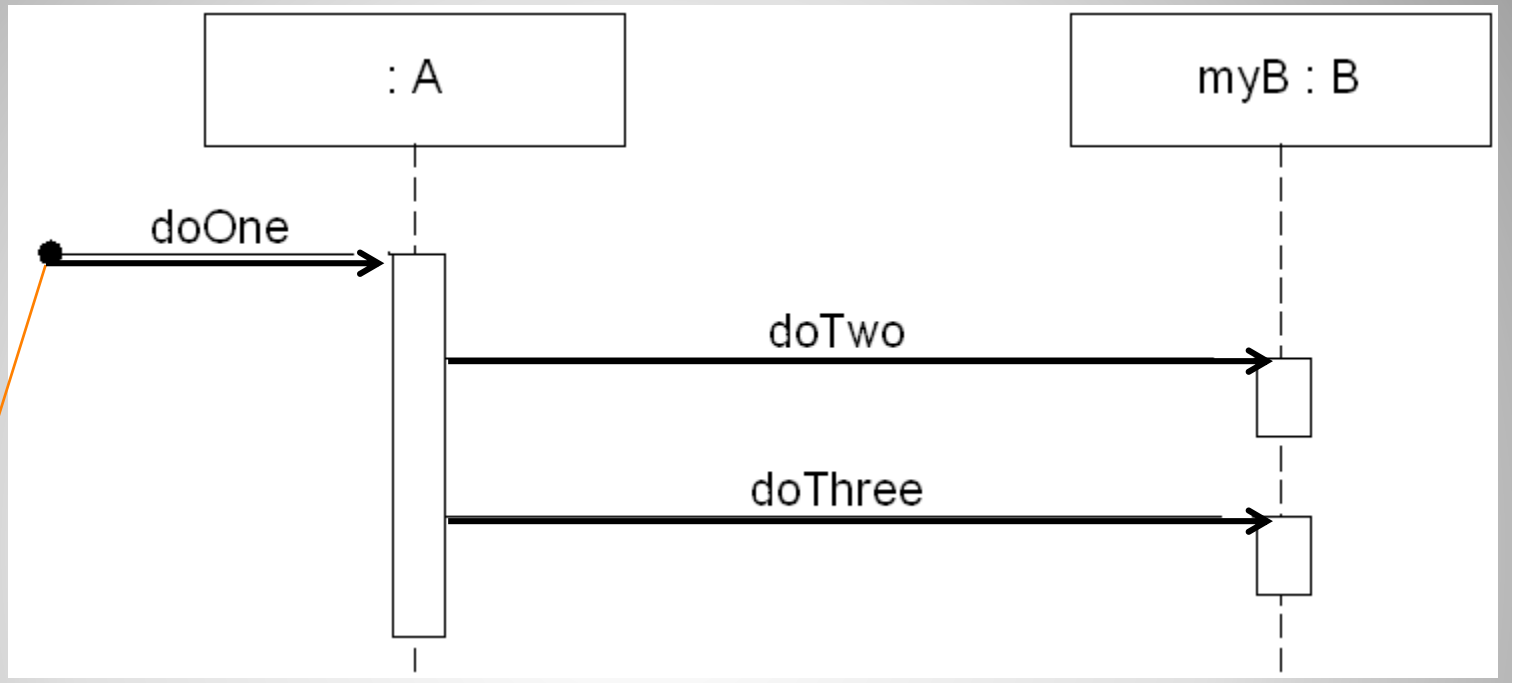

Card Reader
ATM Screen
Joes account : Account
Joe : Operator
1: Insert Card
2: Read Card No
3: Initialize Screen
4: Verify Account
5: Prompt for PIN
6: Enter PIN
7. Validate PIN number
8: [If Validated]: Prompt for Transaction
9: Enter Amount
10:Verify Funds
11. Dispense Cash
12. Eject Card

- When the customer checks in into the hotel the receptionist open the reservation using “Hotel Reservation system”. Reservation system checks the availability of the room and display all the room available in the hotel. The Receptionists then insert the detail of the room i.e., room type, room number and clicks the reservation button to make the room reserve. After reservation, the control returns the confirmation to the system. As soon as the receptionist closes the reservation, the system at the returns to the main menu.

Restaurant Reservation System



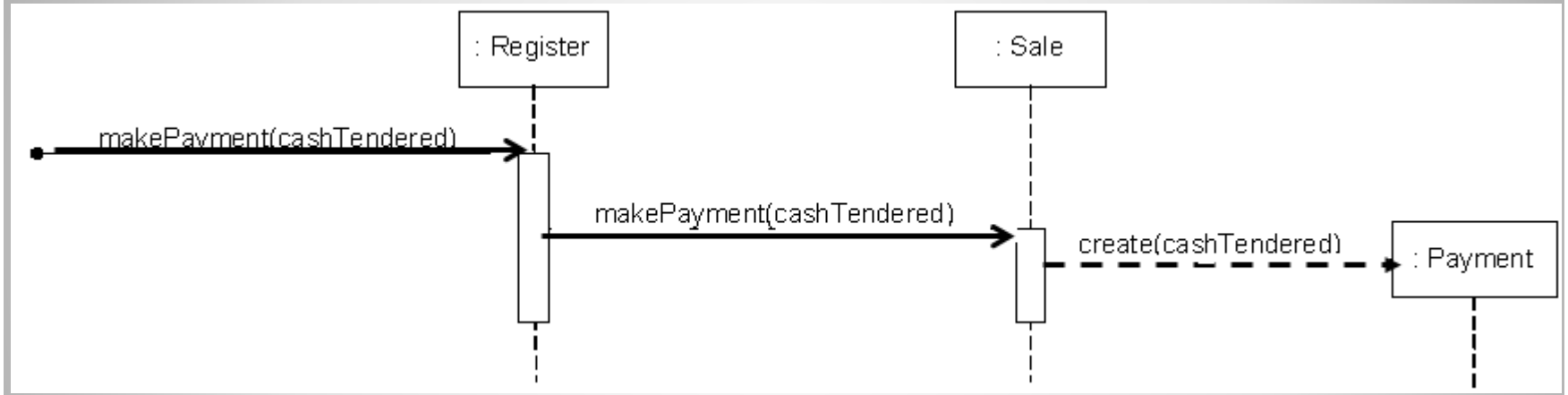
Sequence Diagrams –basic concepts



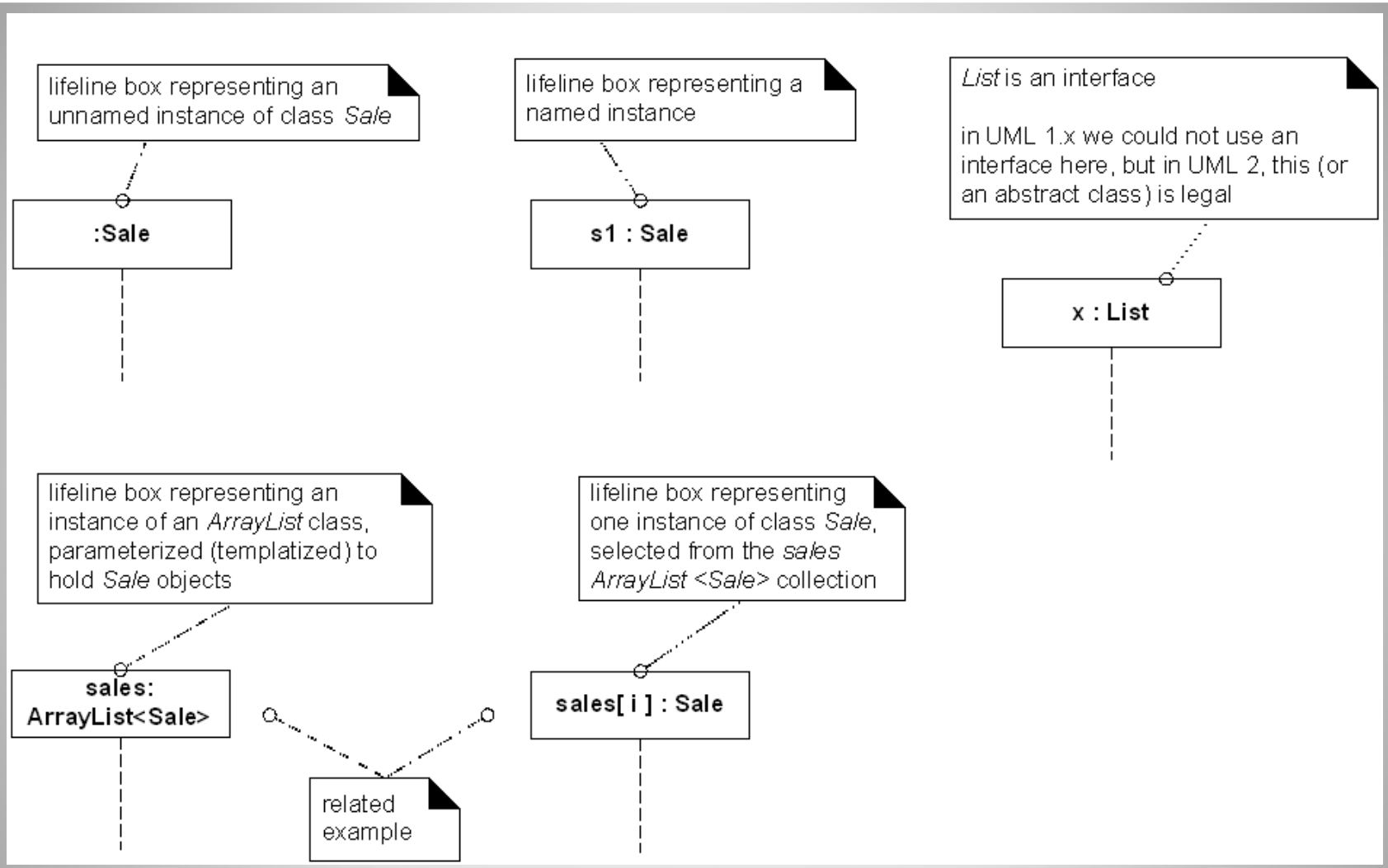
a **found message** whose sender will not be specified

Sequence Diagram

The first message may come from an unspecified participant, and is called a "found message". It is indicated with a ball at the source



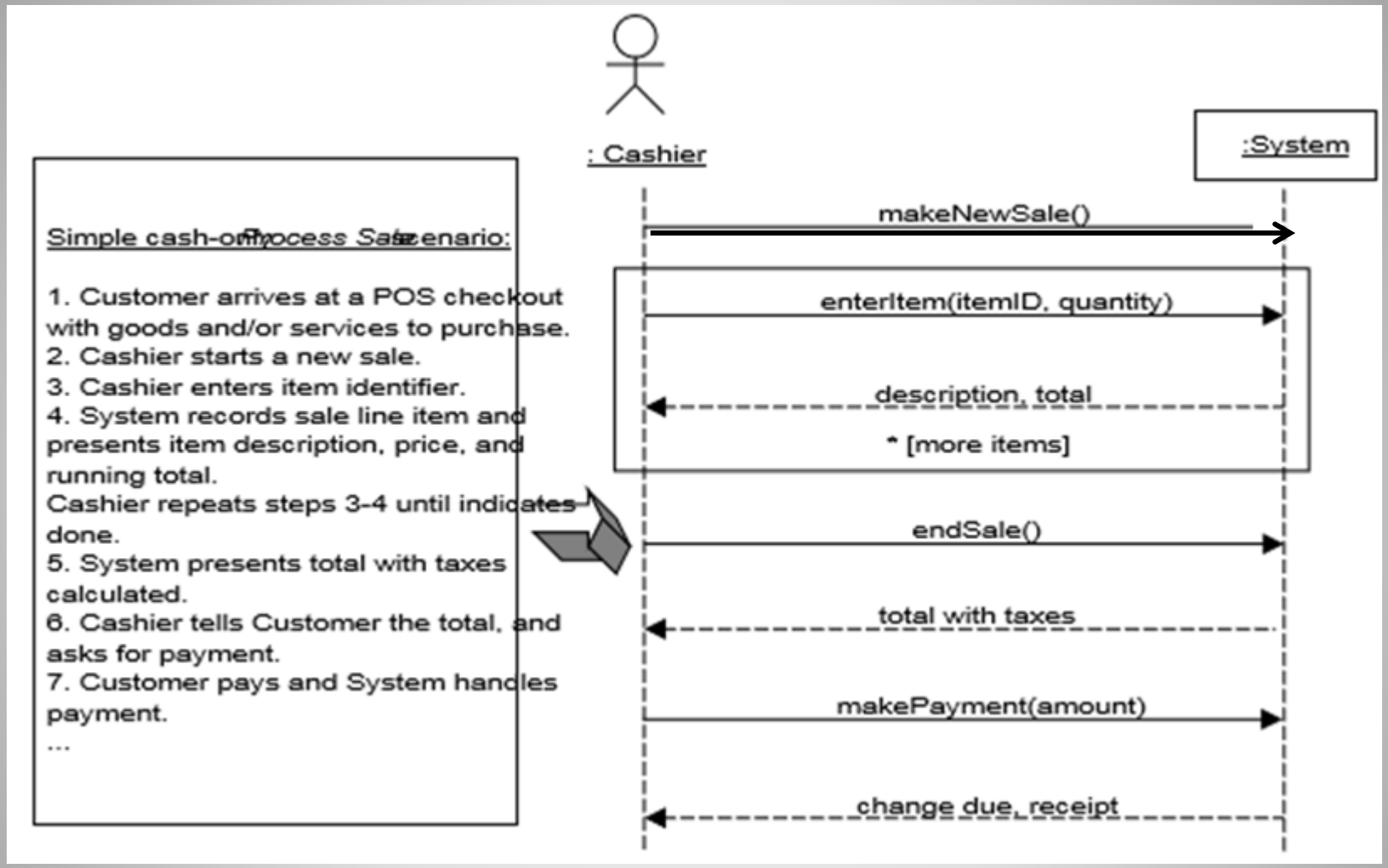
Example: Sequence Diagram: makePayment



Participants with Lifeline Boxes

System sequence Diagram

- A system sequence is a picture that shows one particular scenario of a use case, the events that external actors generate, their order, and inter system events.
- All system are treated a black box.
- The emphasis of the diagram is events that cross the system boundary from actors to system.


Simple cash-only Process Sale scenario:
1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...
: Cashier
:System
makeNewSale()
enterItem(itemID, quantity)
description, total
* [more items]
endSale()
total with taxes
makePayment(amount)
change due, receipt

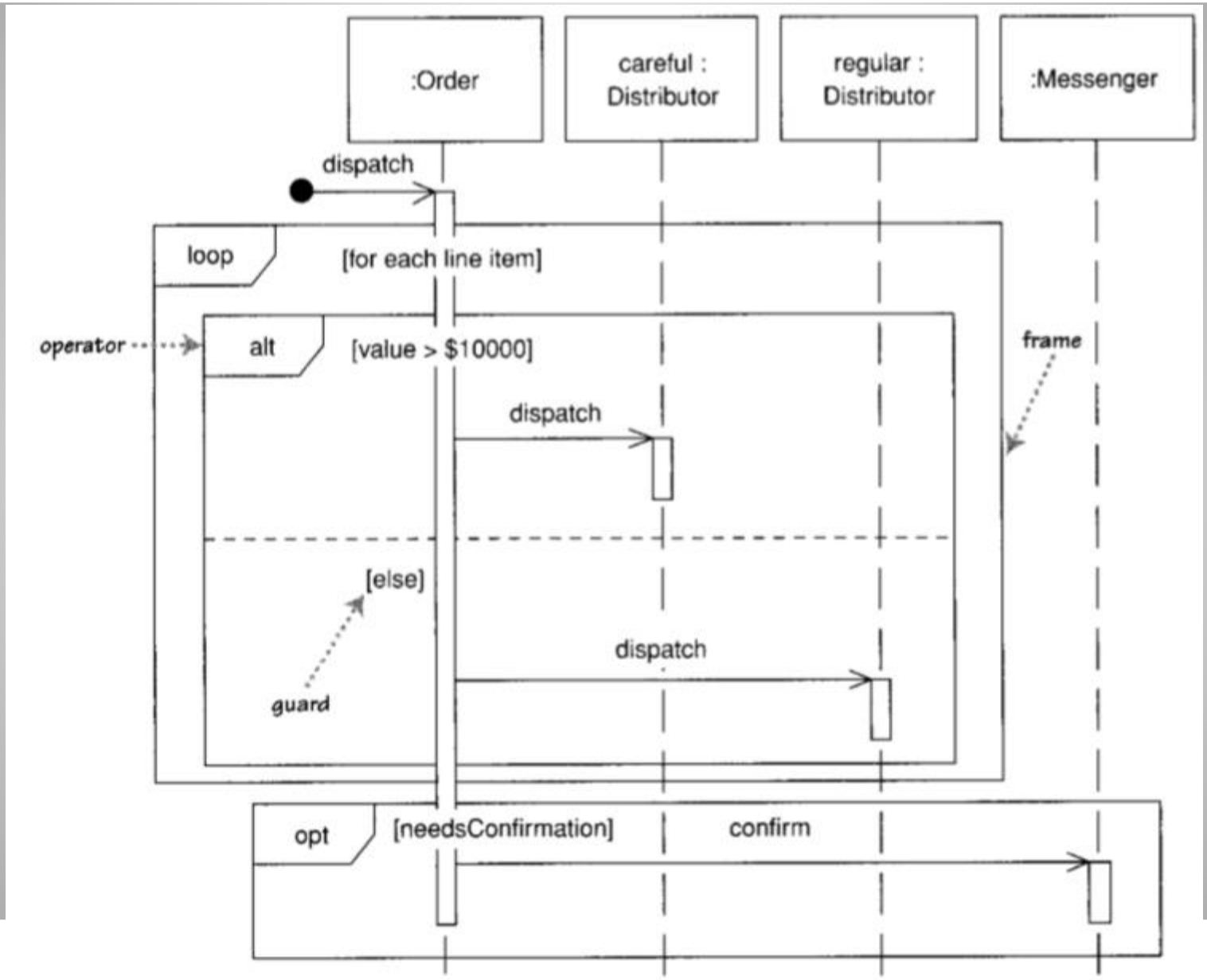
- **Combined fragment** – is a logically consistent area of interaction, a part of a sequence diagram characterized by specific properties defined by the interaction operator such as loop and a guard.

Sequence Diagrams – Diagram Frames

Frame Operators

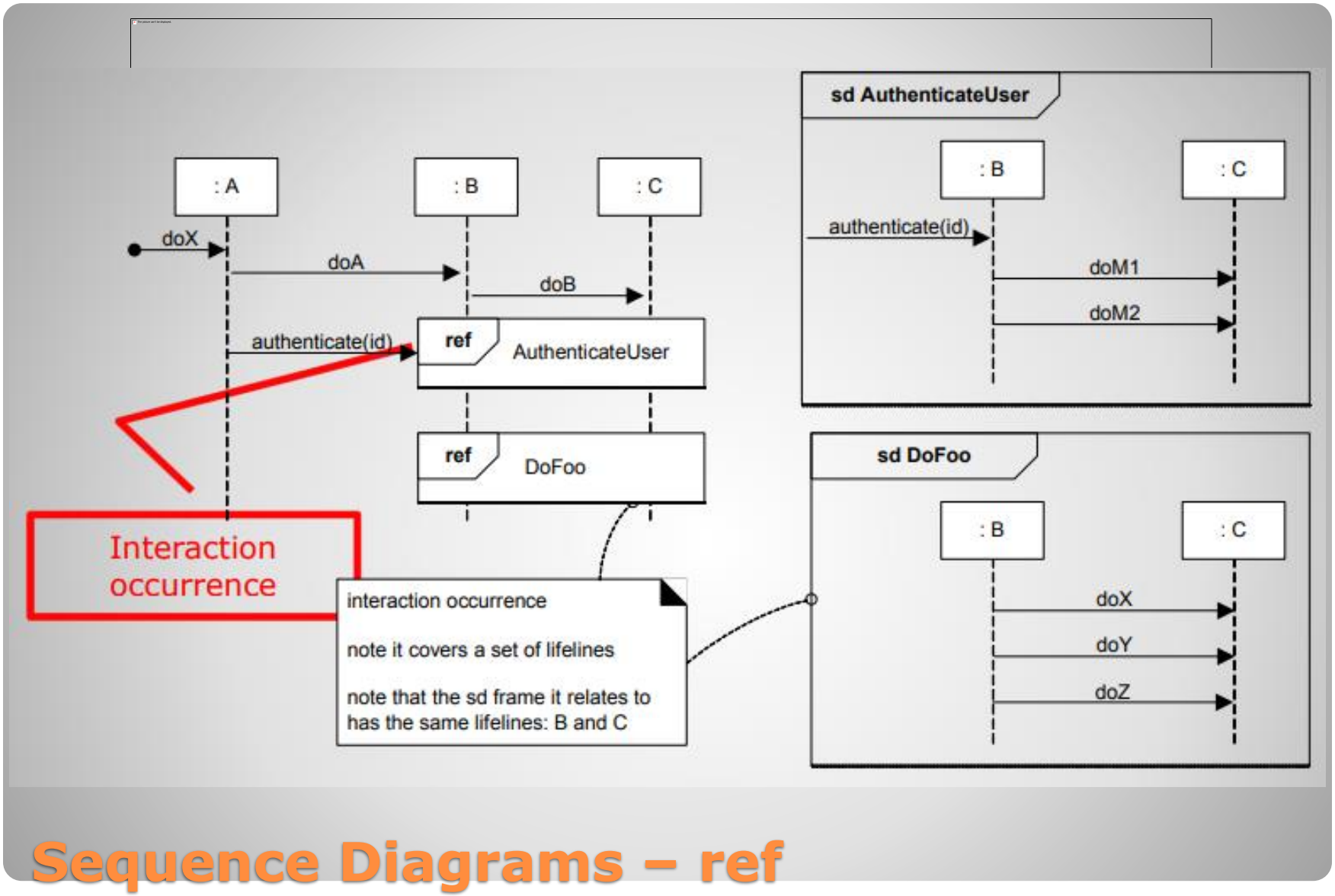
Frame Operator	Description
alt	Alternative fragment for mutual exclusive logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write loop(n) to indicate looping n times.
Opt	Optional fragment that execute if the guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

```
Class Order
Distributor careful=new Distributor
Distributor regular=new Distributor
Messenger messenger=new Messenger
procedure dispatch
foreach (lineitem)
        if (value > $10K)
                careful .dispatch
        else
                regular .dispatch
end if
end for
if (needsConfirmation)
messenger .confirm
end procedure
```


:Order
careful :
Distributor
regular :
Distributor
:Messenger
dispatch
loop
[for each line item]
operator
alt
[value > $10000]
frame
dispatch
[else]
guard
dispatch
opt
[needsConfirmation]
confirm

- **Interaction occurrence** – is a reference to a linked interaction diagram, placed within the base diagram
- Interaction occurrences are especially useful in case of extensive sequence diagrams, which refer to other diagrams defined earlier
- Interaction occurrence can be invoked either by a message or by time factor

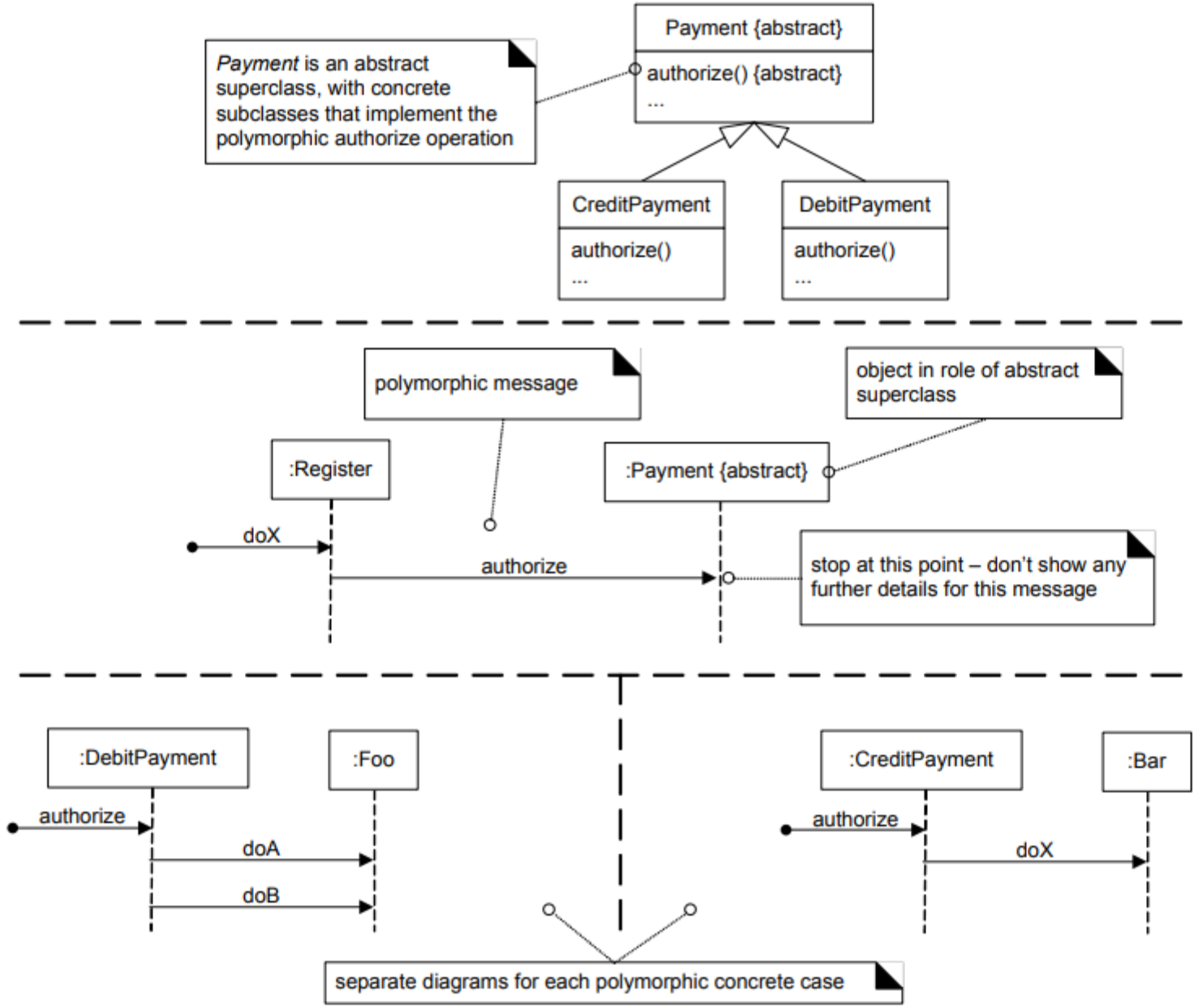
Sequence Diagrams-Interaction Occurrence



Sequence Diagrams – ref

Polymorphic Message

- To show polymorphic message:
  - Use Multiple Sequence Diagram.
    - One that show the polymorphic message to the abstract superclass.
    - Separate sequence diagram detailing each polymorphic case, each starting with found message.


Payment {abstract}
authorize() {abstract}
...
Payment is an abstract superclass, with concrete subclasses that implement the polymorphic authorize operation
CreditPayment
authorize()
...
DebitPayment
authorize()
...
polymorphic message
object in role of abstract superclass
:Register
:Payment {abstract}
doX
authorize
stop at this point – don't show any further details for this message
:DebitPayment
:Foo
authorize
doA
doB
:CreditPayment
:Bar
authorize
doX
separate diagrams for each polymorphic concrete case

- Once a customer checkout, the system checks all required information. If any information is lacking, the system will inform the customer that it needs to be added before the
- order can proceed. The customer sends the required information, after which the system places an order. After placing the order, system verifies the credit card details from the bank. If the credit card info is approved the system inform the customer that order is finalize, other wise the bank inform the system that credit card is not approved and the system convey the message of not approval to the customer. On the basis of which the customer send the new credit card information. The procedure of credit card verification continue until the credit card is not accepted.

Placing an Order (SSD)

- The convener selects a case on the Disbursement GUI (graphical user interface) screen.
- The Disbursement GUI sends the message QueryCase() to the Disbursement Control object, requesting it to query payment-related details about the case.
- The Disbursement Control object services this request by passing a number of messages to the Case object. These include GetPaymentAmount(), GetPcMember(), and GetPcAccount(). These are requests to retrieve payment and Peace Committee member information relevant to the case.
- The convener approves the disbursement for the case.
- The GUI responds to the approval by sending the message CreatePayments() to the Disbursement Control object.
- The Disbursement Control object responds by sending a Create() message to each required Payment object.
- The Payment object sends a Withdraw() message to the cash account and a Deposit() message to the Peace Committee member account.
- The Disbursement Control object finishes the process by sending the message SetPaidStatus() to the Case object to indicate that payments have been made.

Search an Author

- The customer specifies an author on the Search Page and then presses the Search button.
- The system validates the Customer's search criteria.
- The system searches the Catalog for books associated with the specified author.
- When the search is complete, the system displays the search results on the Search Result Page.

- Alternate Course:
  - If the Customer did not enter the name of author before pressing the Search button, the system displays an error message to that effect and prompts the Customer to re-enter an author name.

The scenario begins when the player chooses to start a new round in the UI. The UI asks whether any new players want to join the round; if so, the new players are added using the UI.

All players' hands are emptied into the deck, which is then shuffled. The player left of the dealer supplies an ante bet of the proper amount. Next each player is dealt a hand of two cards from the deck in a round-robin fashion; one card to each player, then the second card.

If the player left of the dealer doesn't have enough money to ante, he/she is removed from the game, and the next player supplies the ante. If that player also cannot afford the ante, this cycle continues until such a player is found or all players are removed.

Poker-Start New Game Round

- The scenario begins after the Start New Round case has completed. The UI asks the first player for a bet. That player chooses to either bet a given amount, or check (no bet). The next player is asked what to do. If the prior player placed a bet, the next player must either match ("see") it, or match it plus add an additional bet ("raise"), or choose not to match and exit the round ("fold"). This continues around the table until an entire pass is made in which all players have either matched all other players' bets or folded. If the next player doesn't have enough money to match the current bet, the player is allowed to bet all of their money. But they can then win only up to the amount they bet; the rest is a "side pot" among the more wealthy players remaining in the round.

Poker-Betting Round

Let's do a sequence diagram for the following casual use case

The scenario begins when the user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time.
The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders.

If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.

Add Calendar Appointment

1. The user presses the “check email” button.
2. The client first sends all unsent email to the server.
3. After receiving an acknowledgement, the client asks the server if there is any new email.
4. If so, it downloads the new email.
5. Next, it deletes old thrashed email from the server.

Check Email