Interaction diagrams

- Interaction diagrams are used to model the dynamic aspects of the system
    - An **interaction diagram is associated with a task** performed by the system or its components
    - Interaction diagrams are determined/built based on activity diagrams and use-case diagrams
    - An interaction diagram generally corresponds to **a use-case** or a functionality
    - The interaction diagram shows how objects and actors communicate together to achieve the task

- Specifically, an interaction diagram allows to describe in detail **the algorithms** in the system
- Interaction diagrams can be subsequently used in **the implementation of class methods**

Interaction diagrams

- The essential elements of an interaction diagrams
  - Objects
  - Actors
  - Messages

- Actions between objects and actors are
  - message sendings
  - object creations and destructions

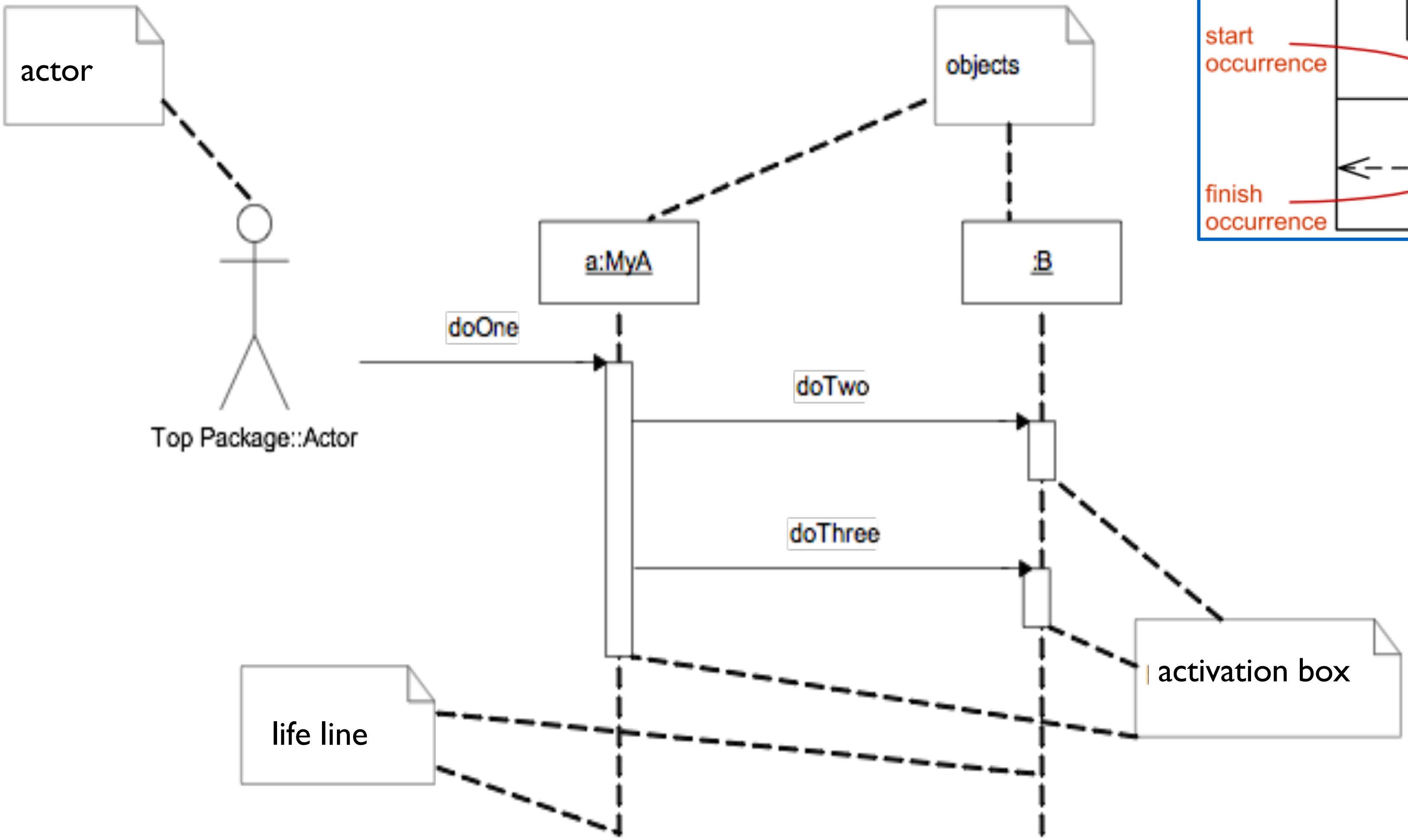
- **Two types of interaction diagrams**
  - **Sequence diagrams**
    - The temporal sequence of interactions
  - **Collaboration diagrams**
    - An instance of class diagram

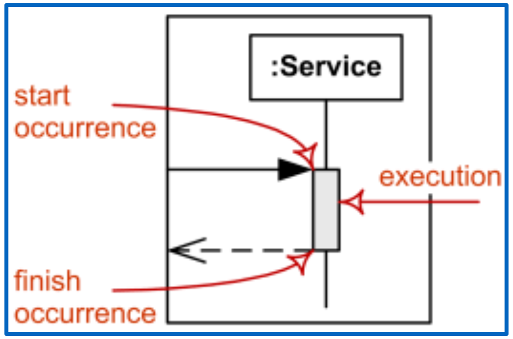
Sequence diagram

- A **sequence diagram** describes the **temporal sequence** of exchanges of messages between objects and the actor to perform a certain task
  - The **actor** who initiates interactions is usually found on the far left
  - The **objects** are placed horizontally on the diagram
  - The vertical dimension represents time
  - Each object or actor is associated with a **life line** representing the time where the object or actor is
  - An **activation box** represents the object activation period

Sequence diagram

- Notation





Sequence diagram

- Messages
  - Message is the medium of communication between objects
  - The general form of message

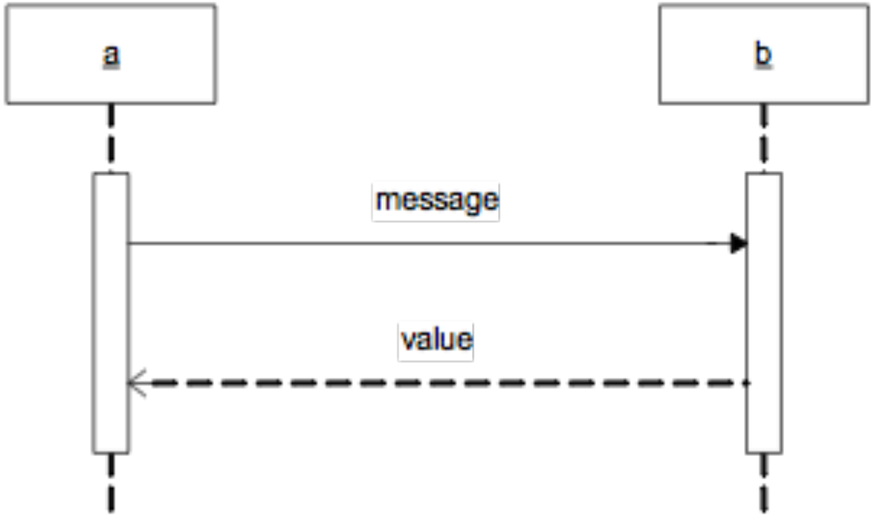
    [guard]message(parameters)

    - **guard**: a condition must be satisfied in order to send the message
    - **message**: the identifier of the sent message
    - **paramaters**: a list of parameter values

    - Note: guard and parameters can be omitted

Sequence diagram

- The return values
  - Sending a message to an object cause **the execution of a method** of this object
    - This method can optionally return a value
  - The return values may be omitted or be explicitly described
    - either as the following form
      [guard]value := message(parameters)
    - or by a return message that represents graphically

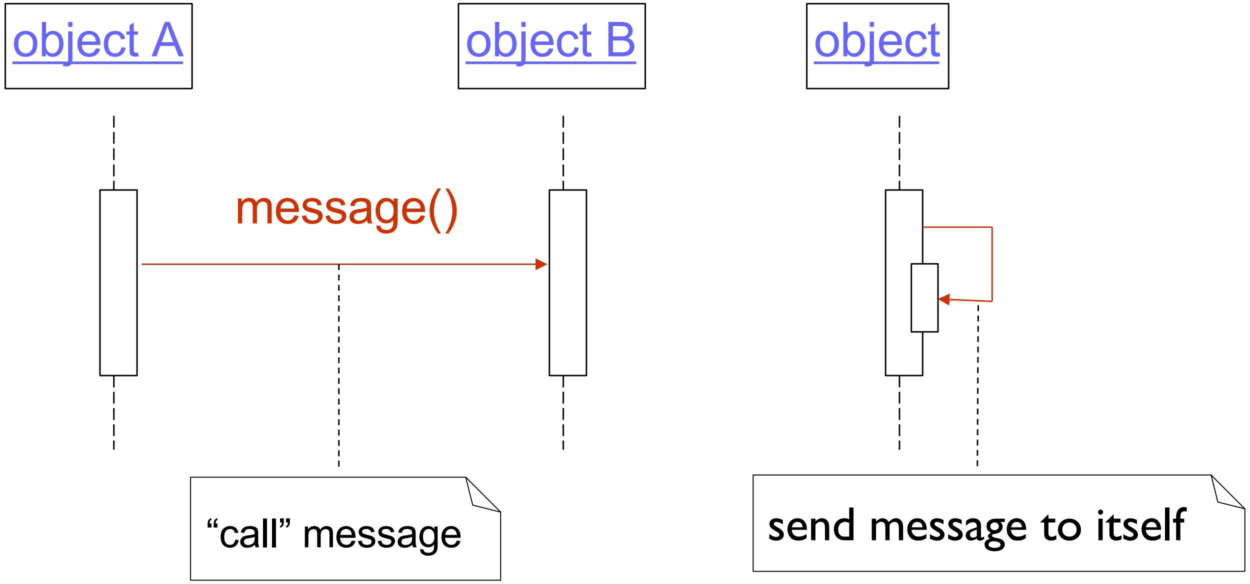


Sequence diagram

- Types of message
  - “call” message
  - “return” message
  - “send” message
  - “create” message
  - “destroy” message

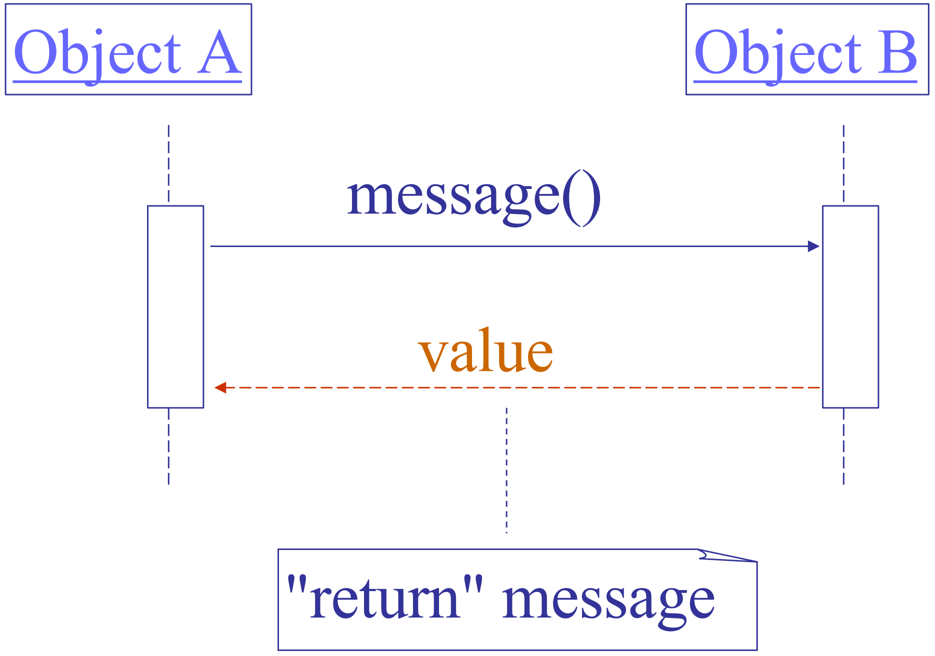
Sequence diagram

- “call” message
  - A “call” message invokes an operation/method of the object
  - A “call” message is a **synchronous message**: the object that sends the message must wait for the termination of the execution of the message before doing other tasks
  - An object can send message to itself
  - Notation



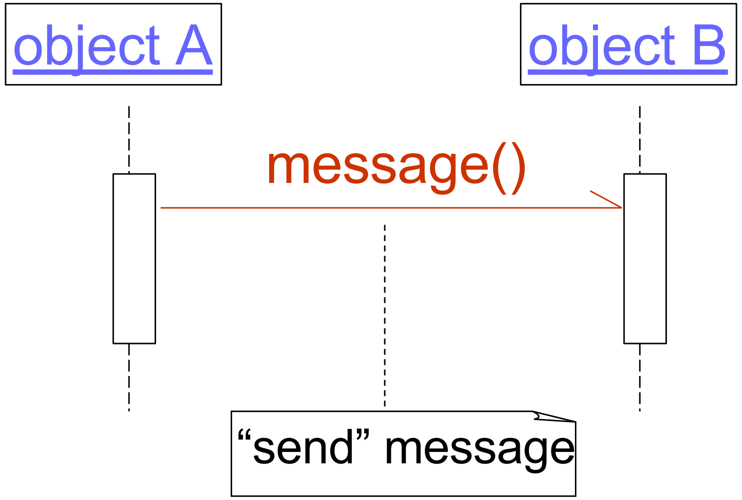
Sequence diagram

- □ The “return” message returns a value for the calling object
- □ Notation



Sequence diagram

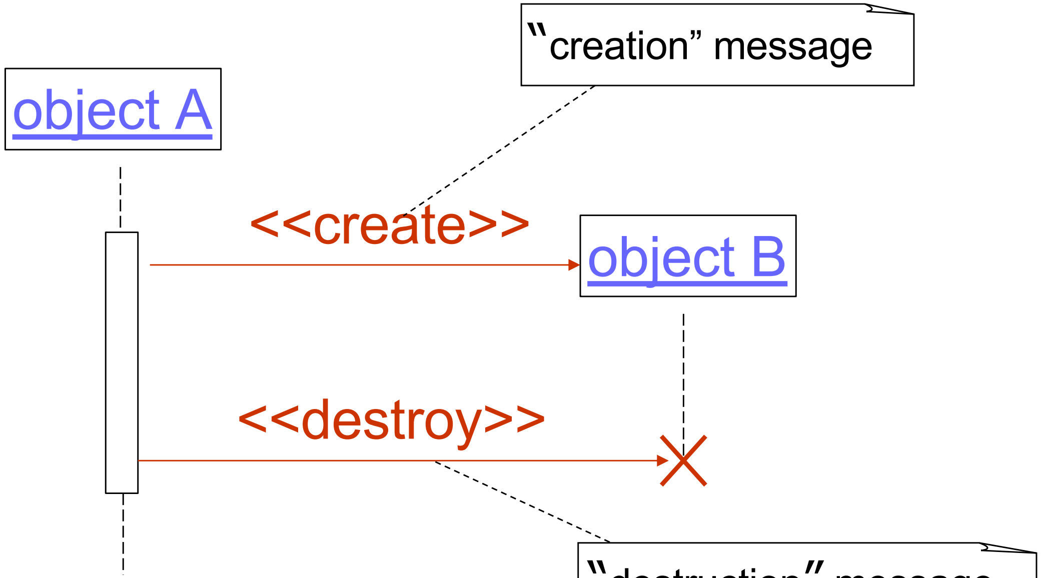
- “send” message
  - A “send” message sends a signal to an object
  - A “send” message is an **asynchronous message**: once the object sends the message, it expects nothing and continues to do other tasks
  - Notation



  - Asynchronous message is often used in multi-threaded environment
    - For example, *Thread.start(), Runnable.run()* in Java

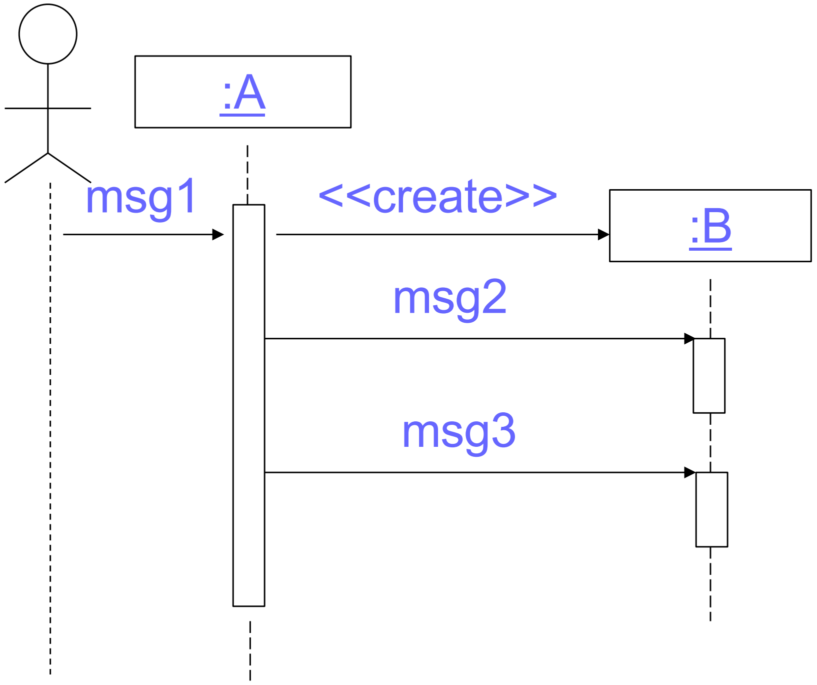
Sequence diagram

- “creation” message
  - invokes the creation method of object (constructor)
- “destruction” message
  - invokes the destruction message of message (destructor)
- Notation



Sequence diagram

- □ Example
  - ■ The sequence diagram and the corresponding code

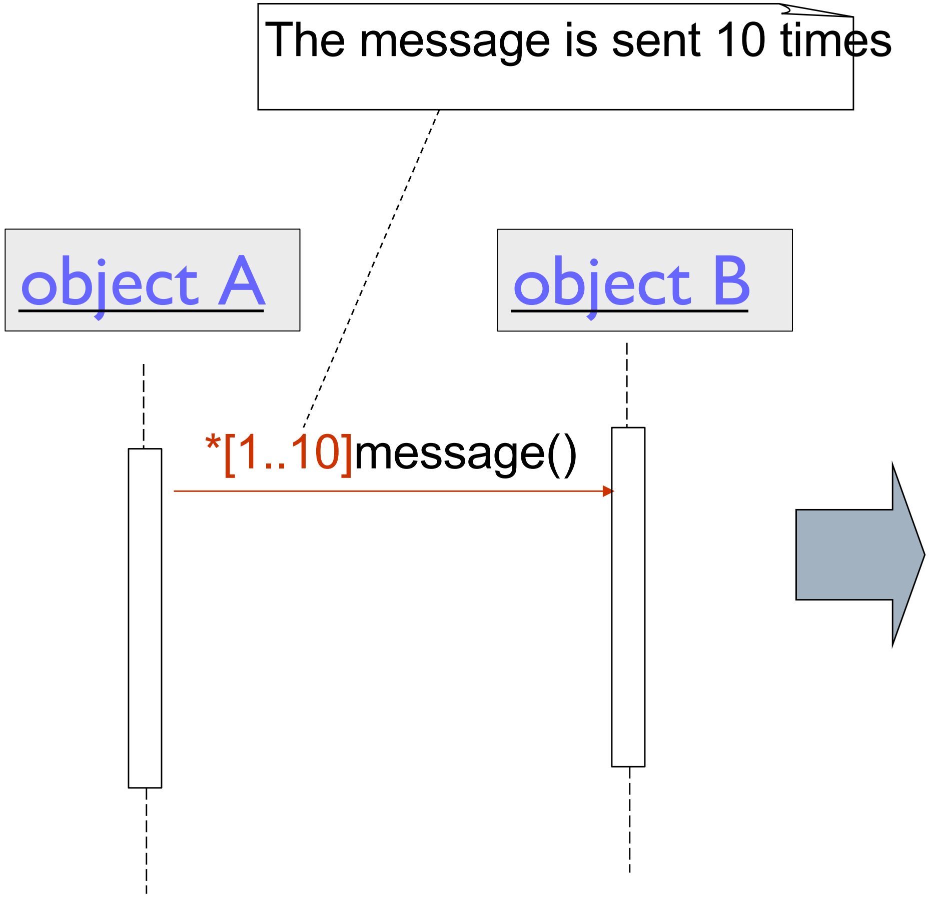


```
public class A
{
  private B objB;
  public void msg1()
  {
      objB = new B();
      objB.msg2();
      objB.msg3();
   }
}

public class B
{
  …
   public void msg2() { … }
   public void msg3() { … }
}
```

Sequence diagram

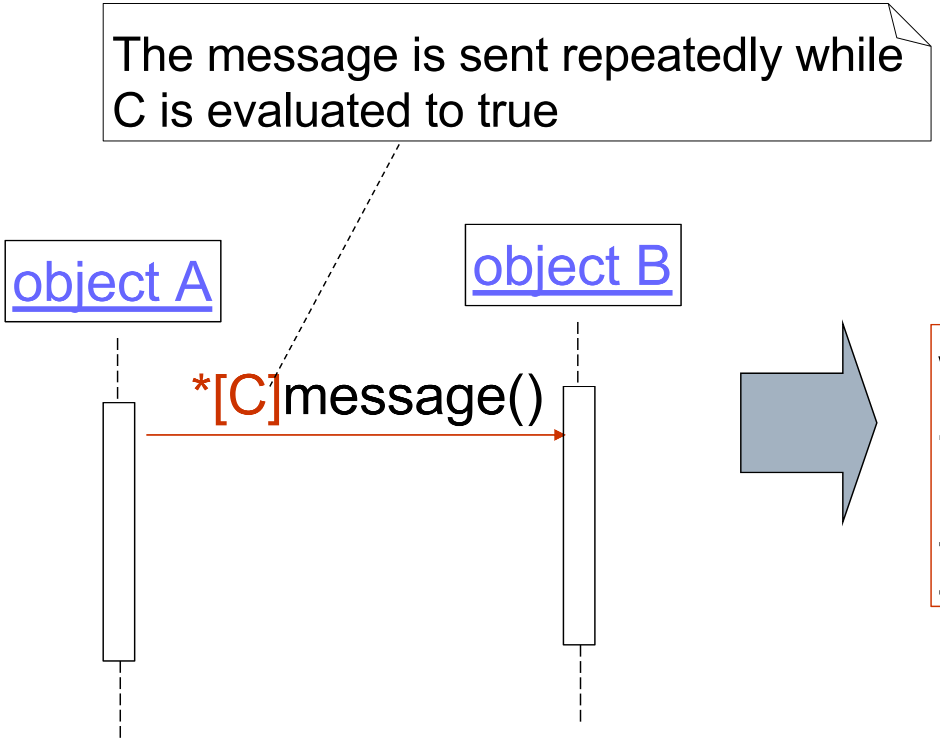
- A message can be **sent iteratively**
- Example



```
for(i = 1; i<= 10; i++)
{
        objectB.message()
}
```

Sequence diagram

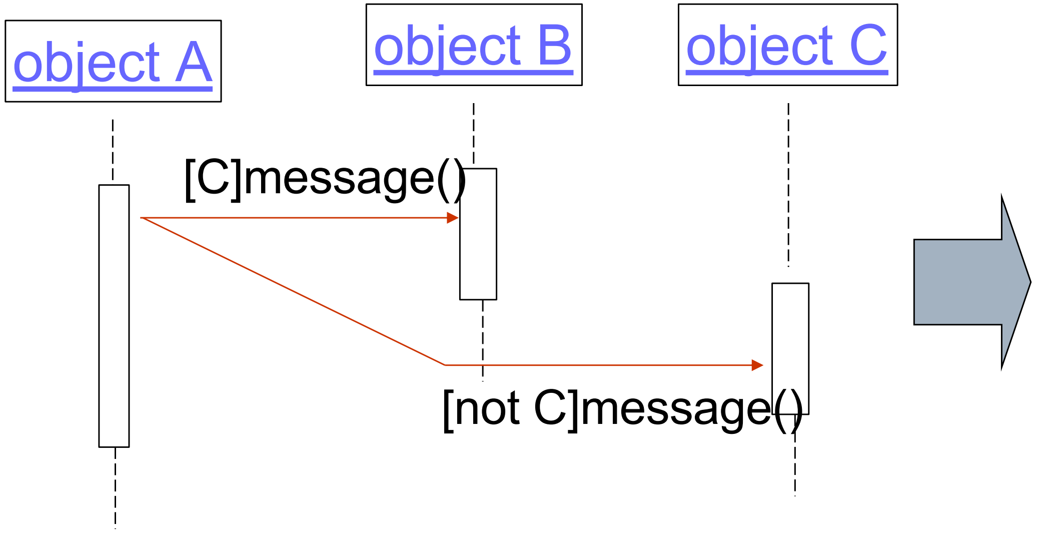
- □ A message can be sent iteratively based on a condition
- □ Example



```
while(C)
{
        objectB.message()
}
```

Sequence diagram

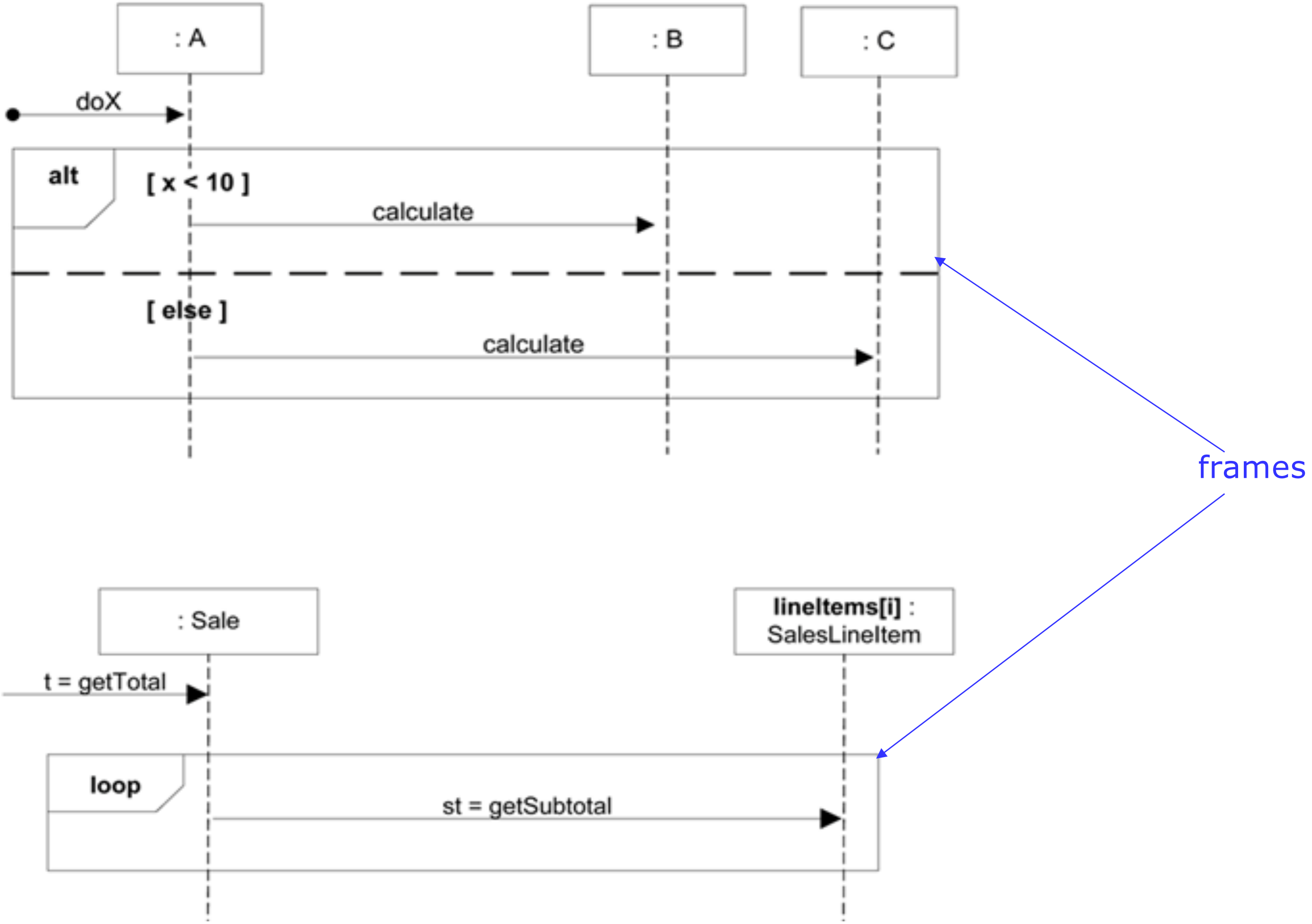
- □ The sending of a message can depend on a **decision**
- □ Example



```
if(C)
    objectB.message();
else
   objectC.message();
```

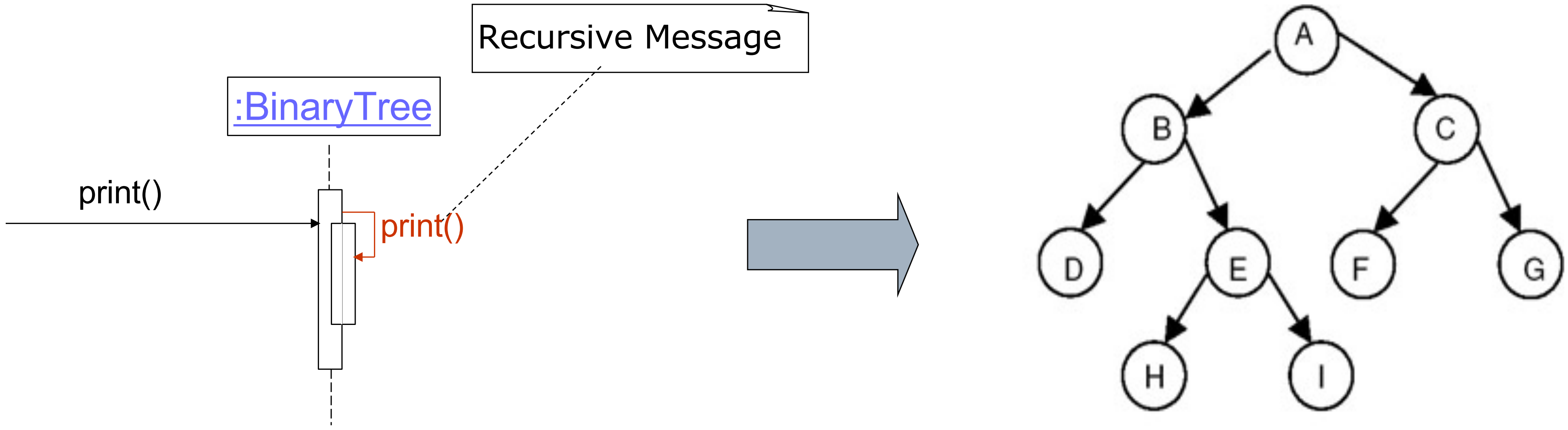
Sequence diagram

- □ Note: UML 2.x notations allow the use of frames to represent the conditions or iterations



Sequence diagram

- A message can be called **recursively**
- Notation



Inorder : DBHEIAFCG

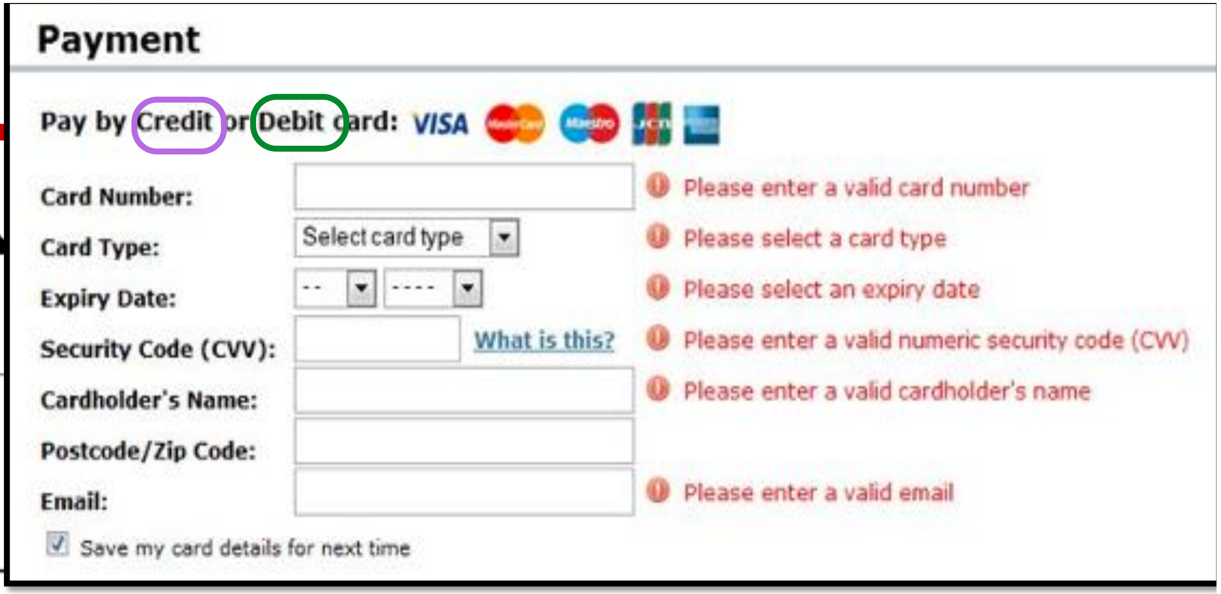
Preorder : ABDEHICFG

Postorder : DHIEBFGCA

Sequence diagram

Modelling a polymorphic message

Payment is an abstract superclass, with concrete subclasses that implement the polymorphic authorize operation

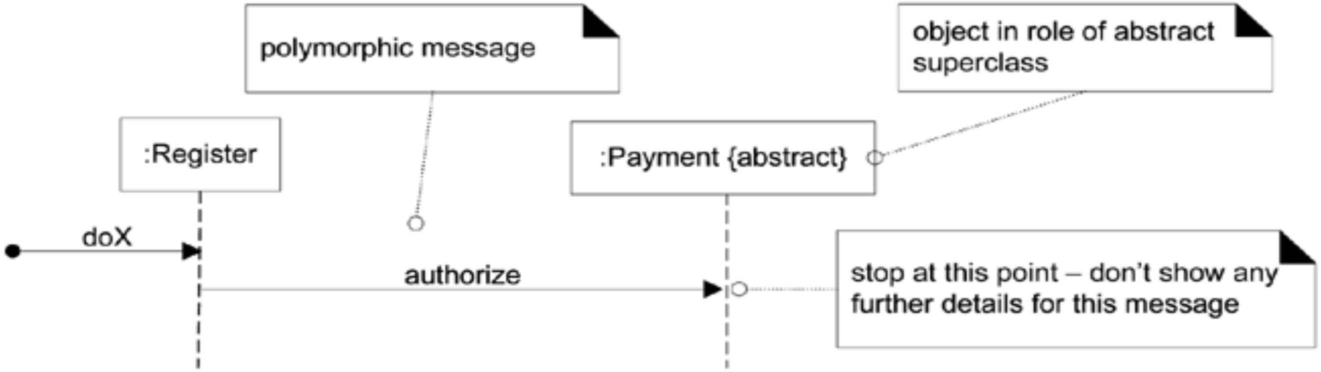


Payment

Pay by Credit or Debit card: VISA

Card Number:		Please enter a valid card number
Card Type:	Select card type	Please select a card type
Expiry Date:	-- ----	Please select an expiry date
Security Code (CVV):	What is this?	Please enter a valid numeric security code (CVV)
Cardholder's Name:		Please enter a valid cardholder's name
Postcode/Zip Code:		
Email:		Please enter a valid email

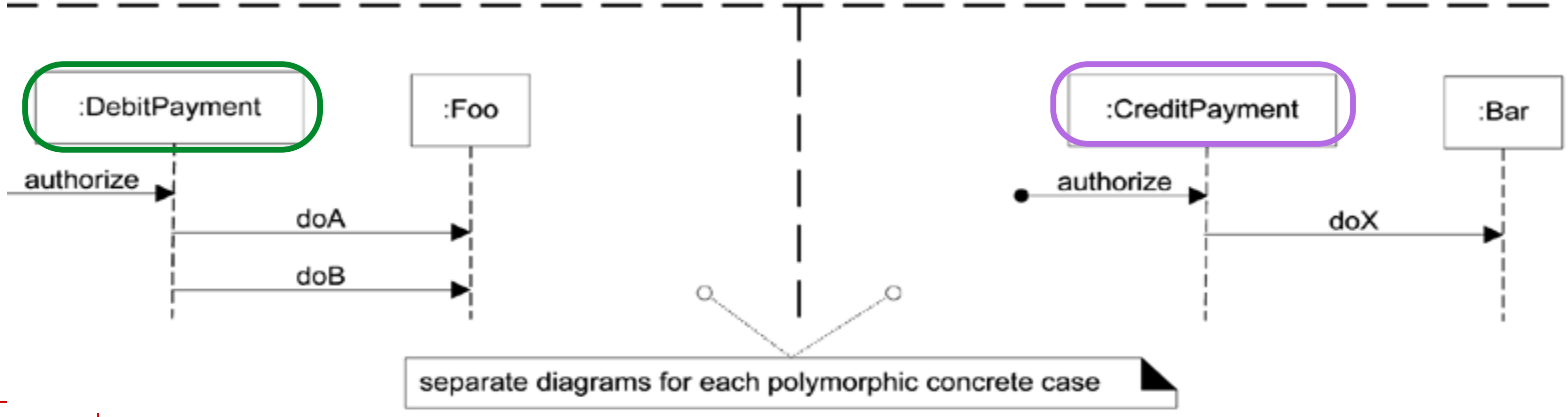
☑ Save my card details for next time



polymorphic message

object in role of abstract superclass

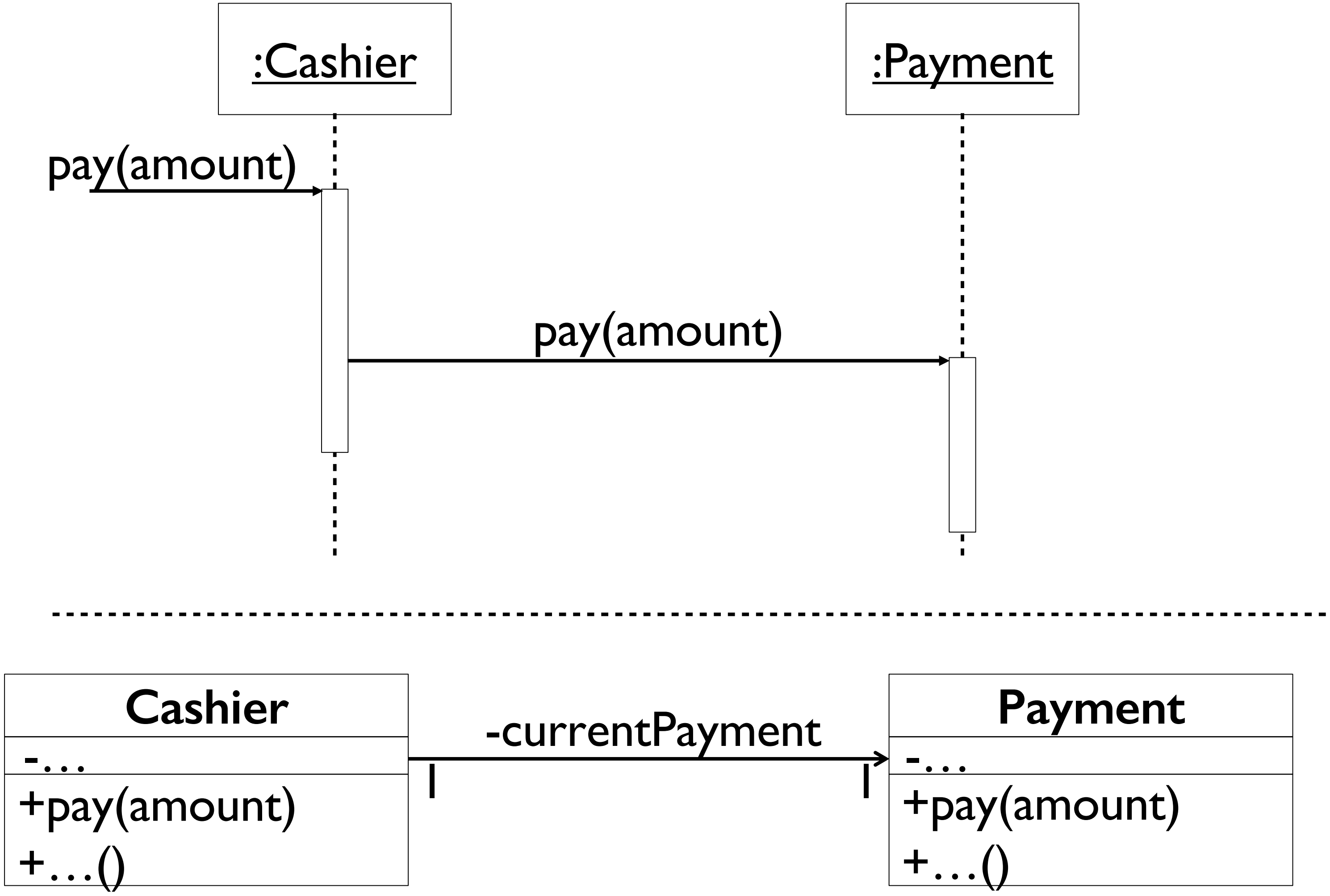
stop at this point – don't show any further details for this message



separate diagrams for each polymorphic concrete case

Sequence diagram

- Relationship between class diagram and sequence diagram



Sequence diagram from use-case

Basic Course

The Customer specifies an author on the Search Page and then presses the Search button.

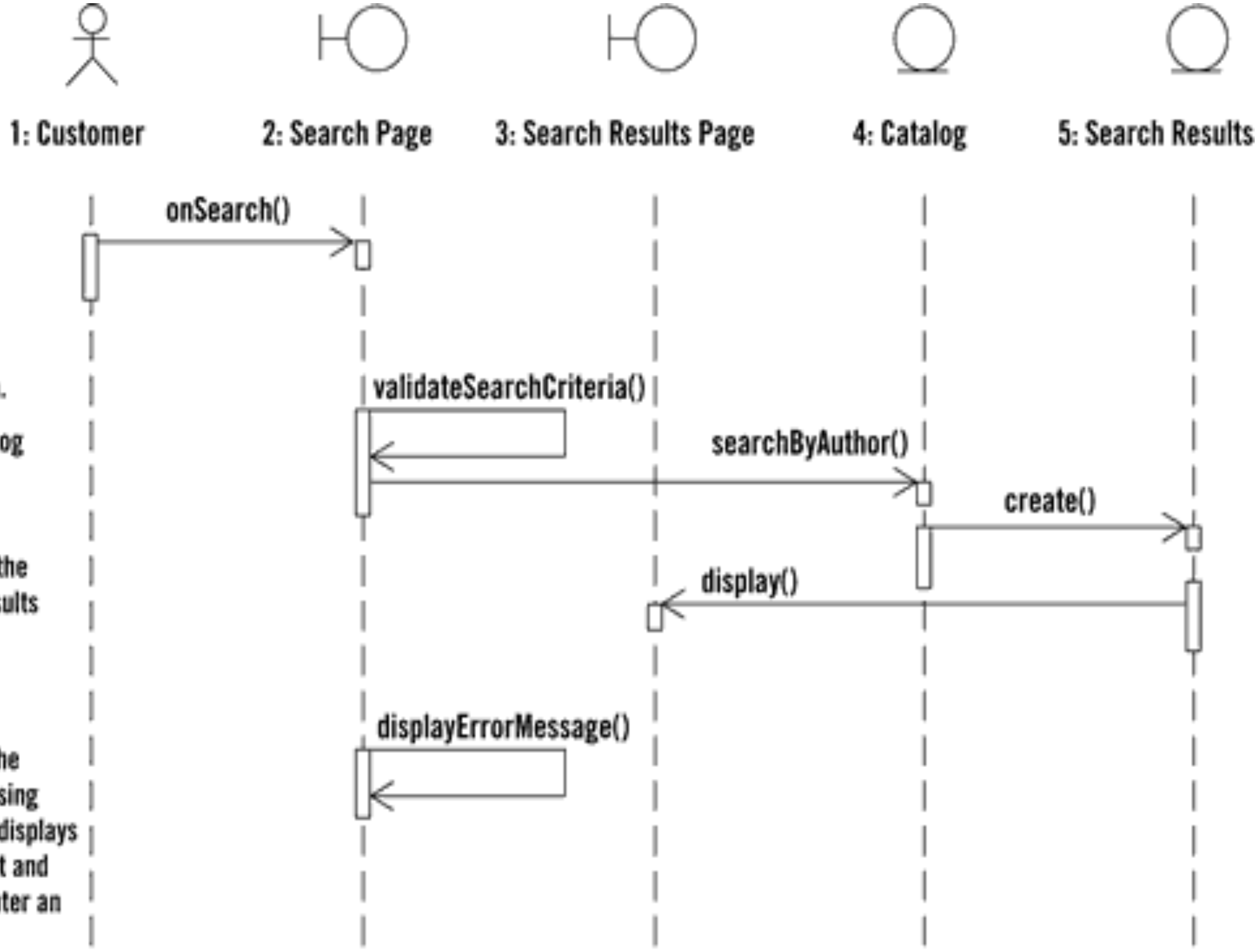
The system validates the Customer's search criteria.

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 Results Page.

Alternate Course

If the Customer did not enter the name of an 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.

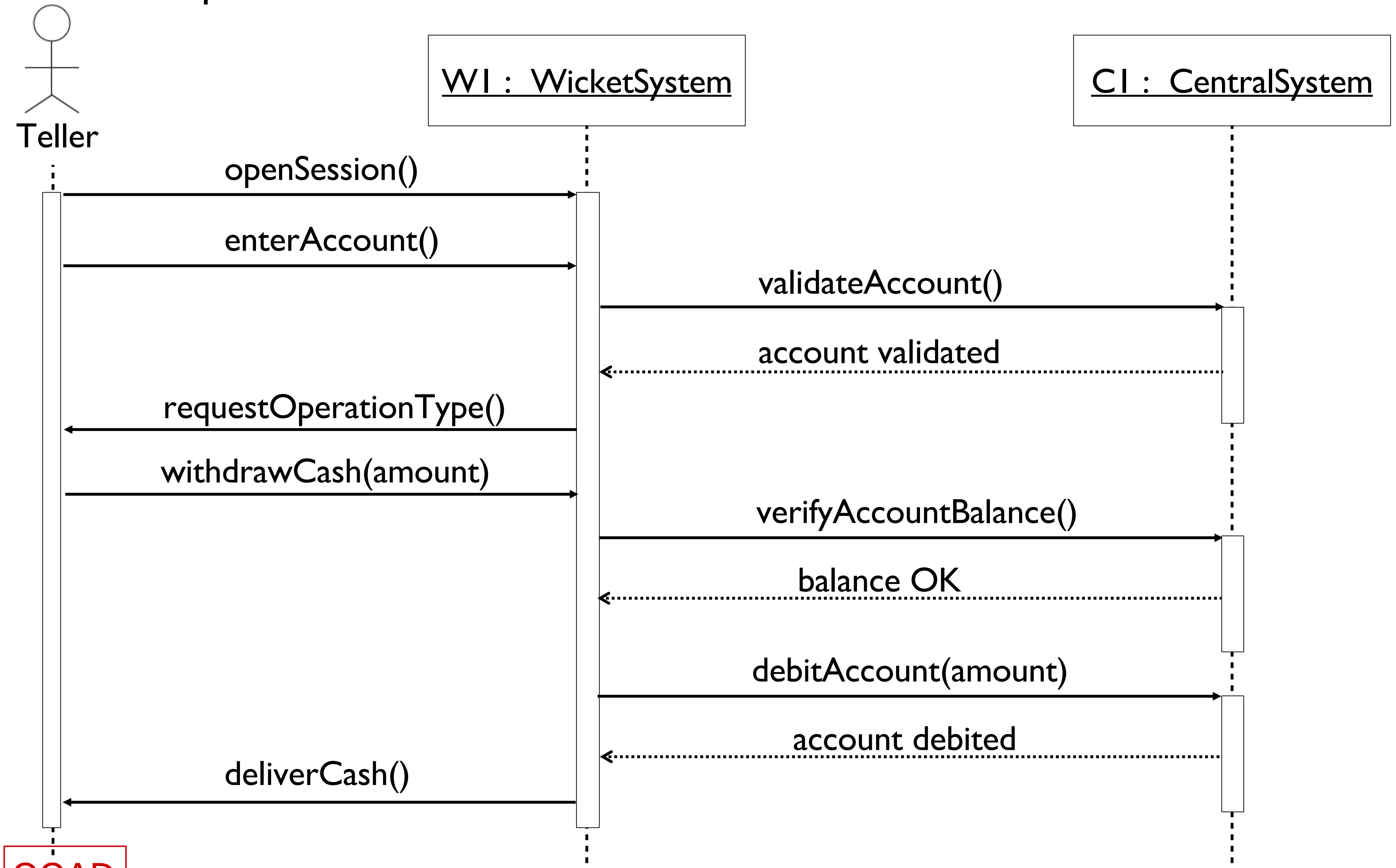


Sequence diagram

- □ Example: Cash withdrawal at the bank

Sequence diagram

- Example: Cash withdrawal at the bank

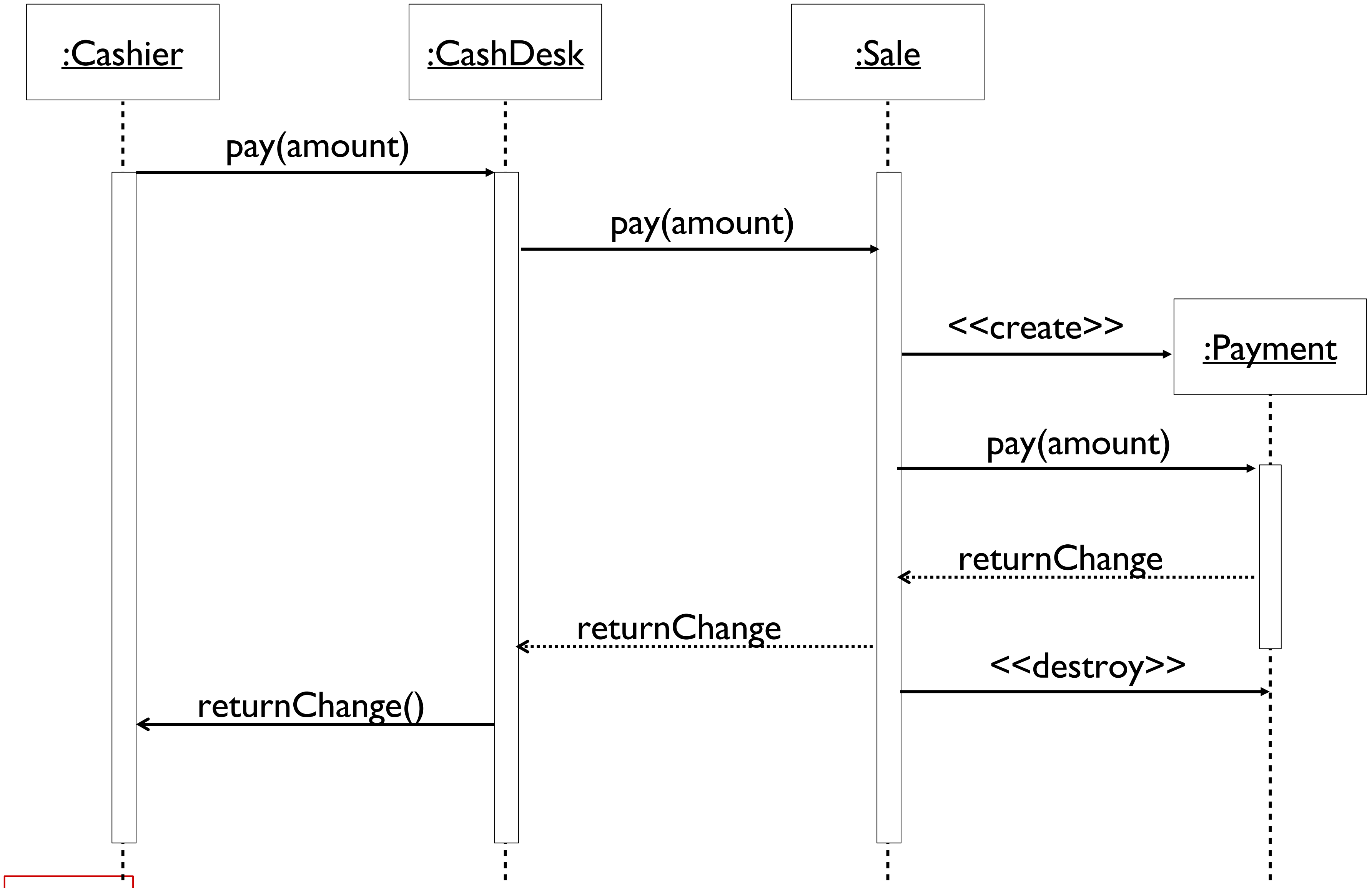


Sequence diagram

- Example: Use-case “cash payment”

Sequence diagram

- Example: Use-case “cash payment”



Sequence diagram

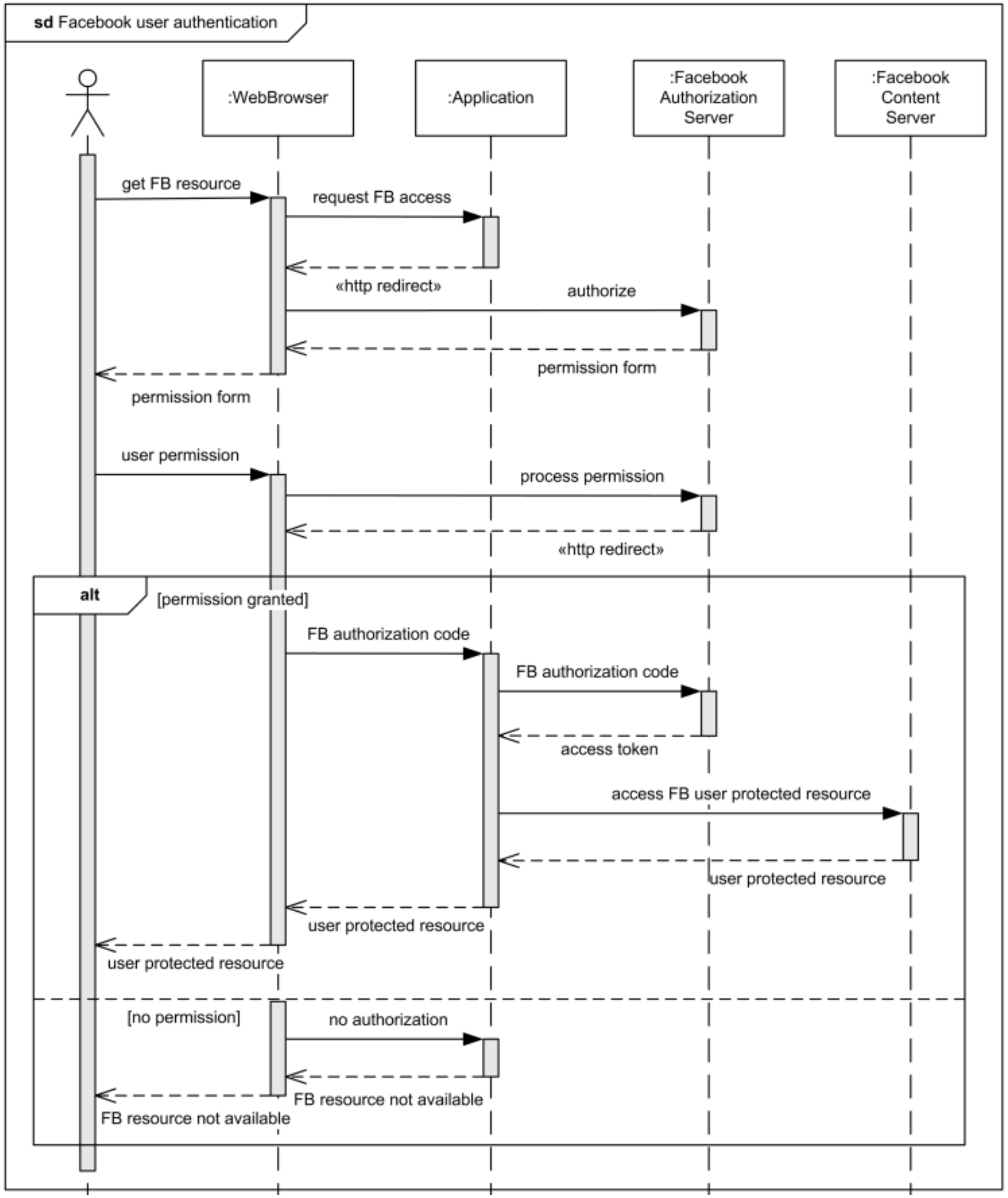
- □ Example:

Facebook

Web

User

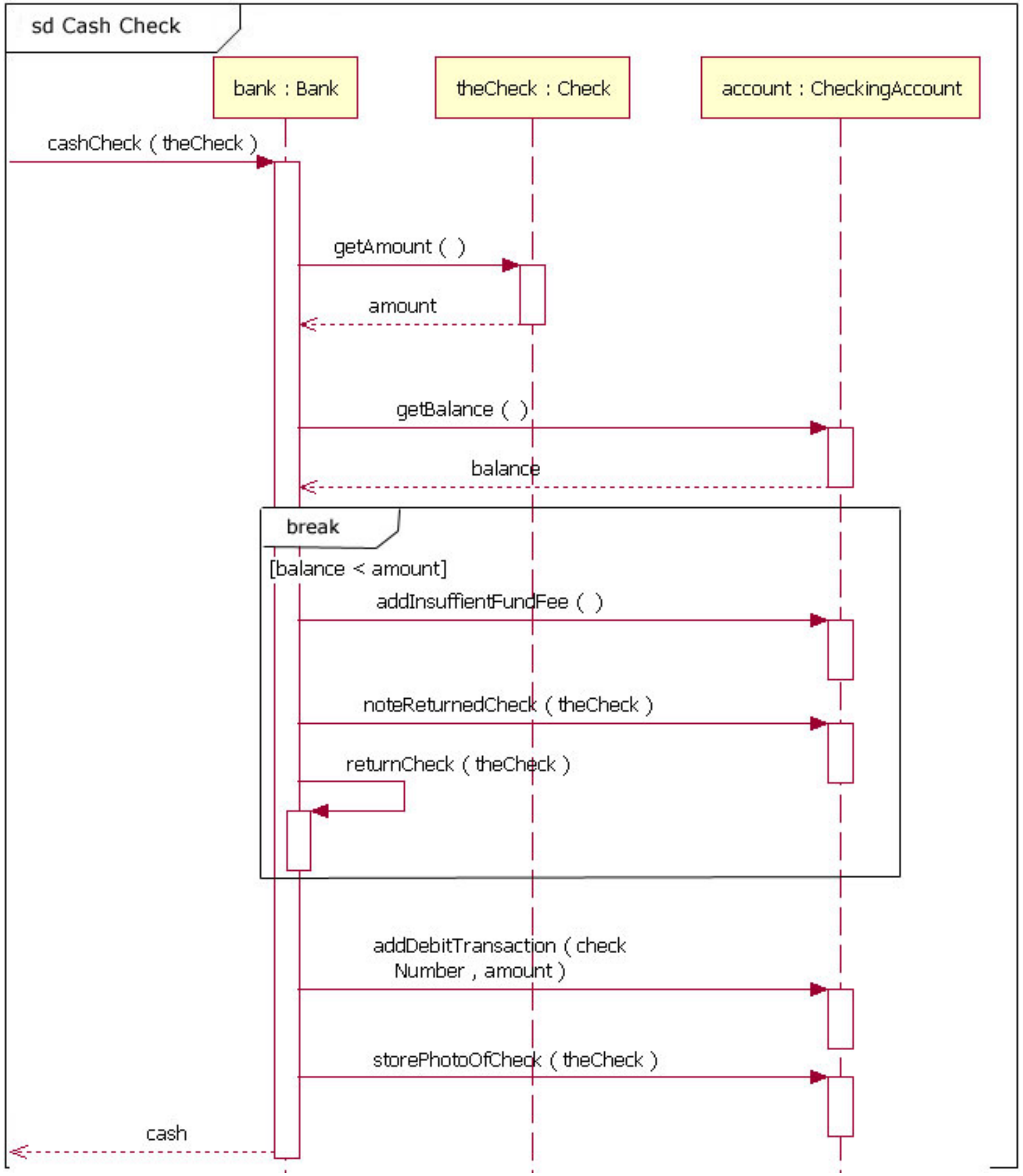
Authentication

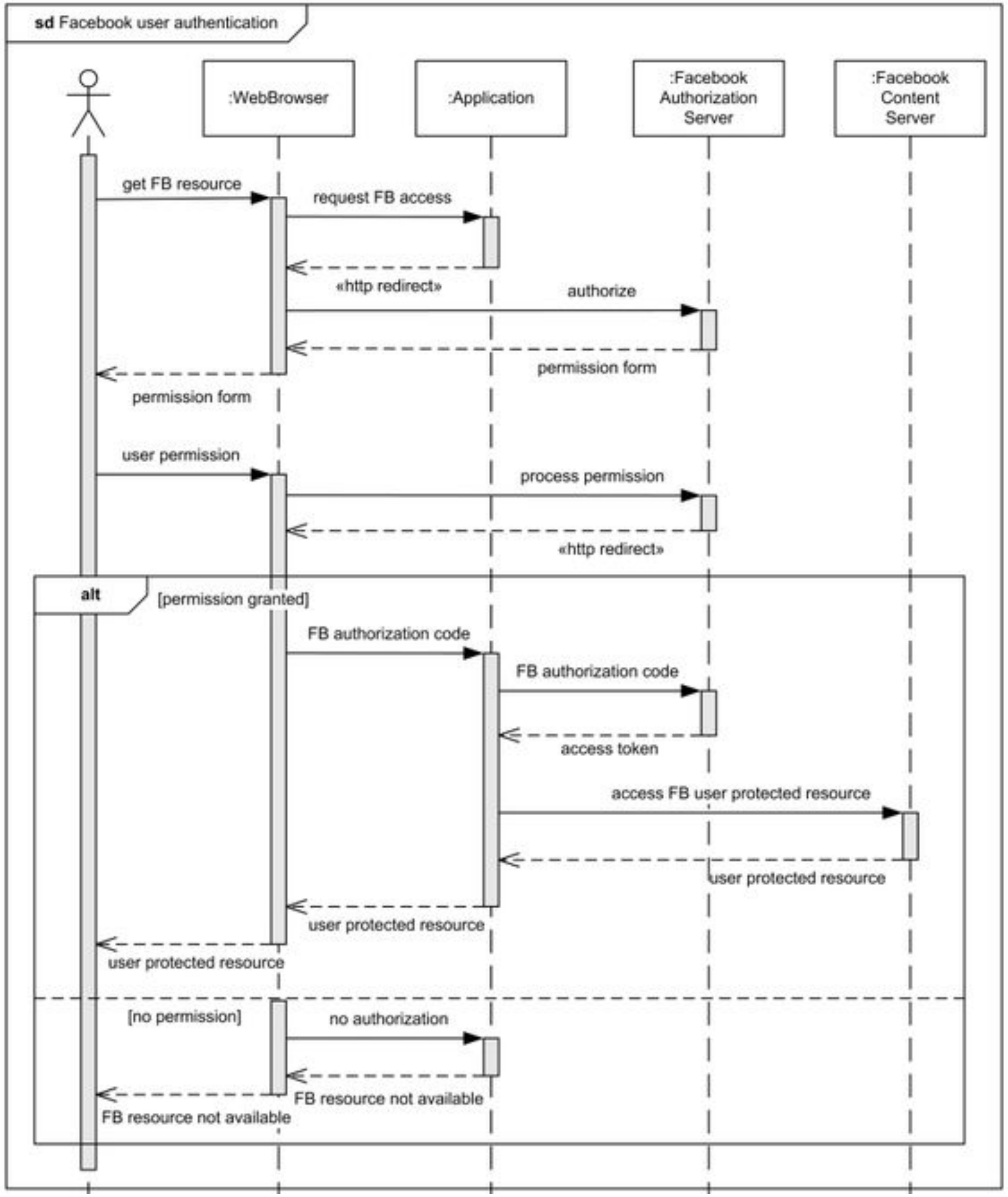


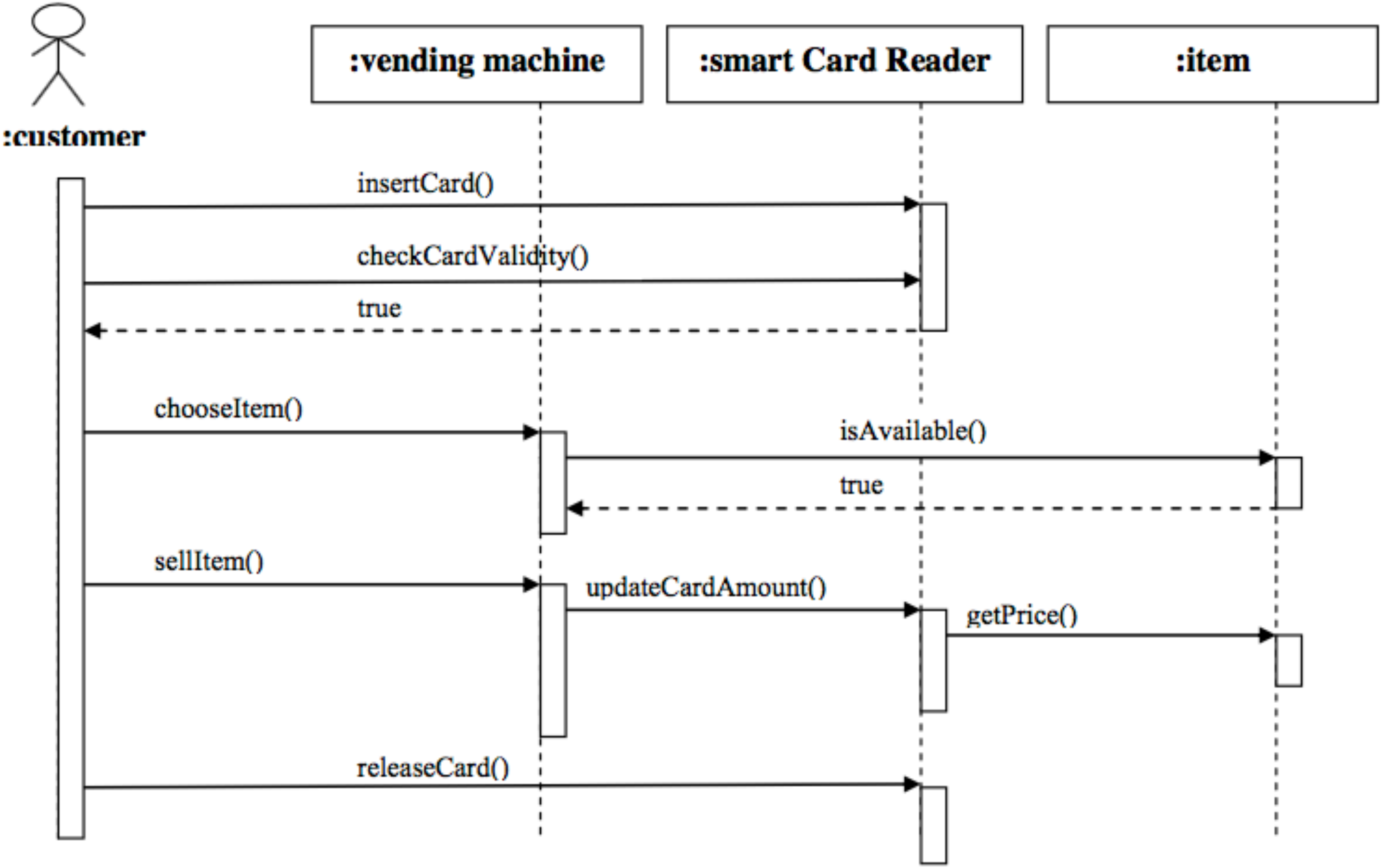
Sequence diagram

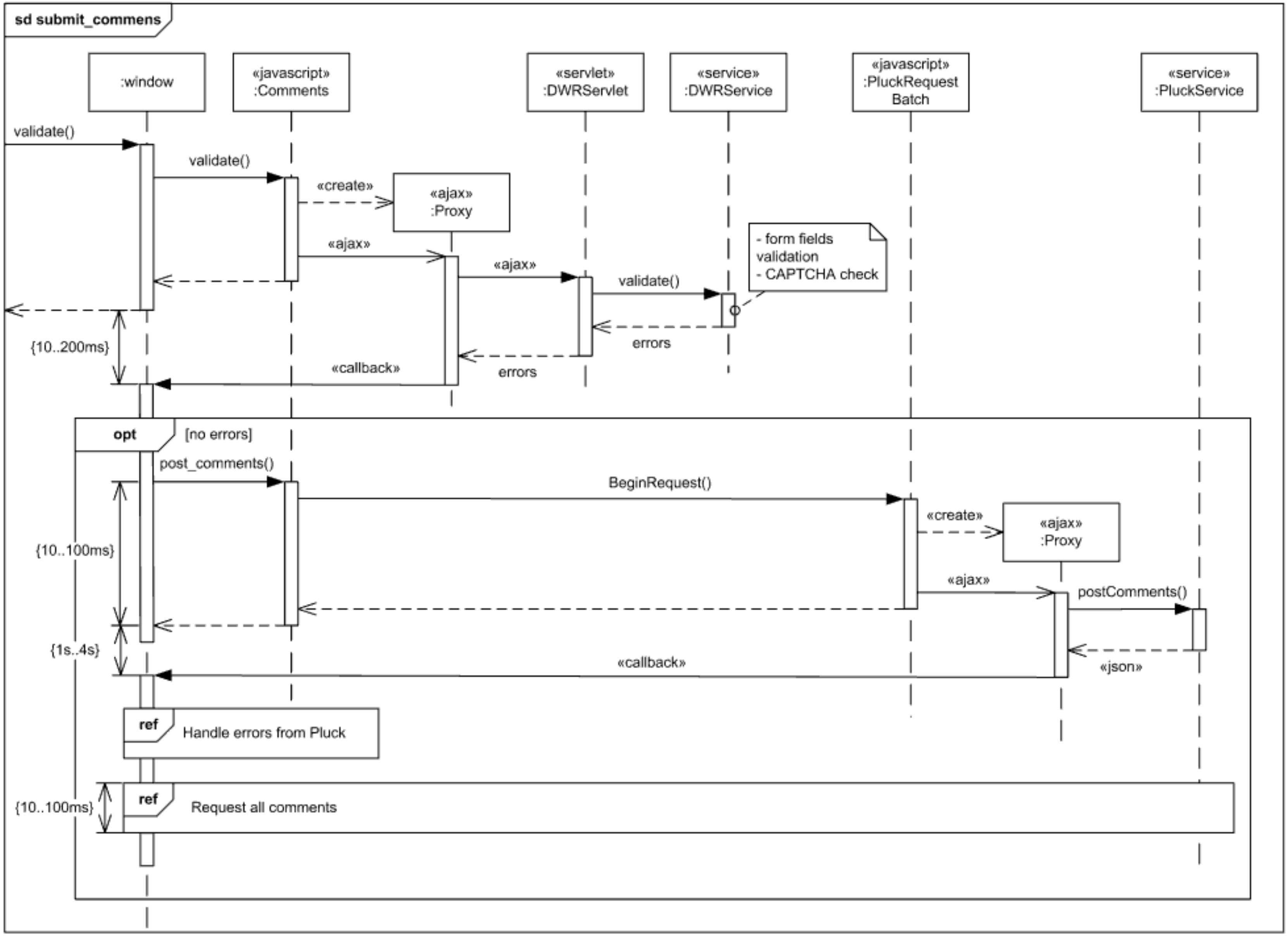
- Example:

Cash Check




sd Facebook user authentication
:WebBrowser
:Application
:Facebook Authorization Server
:Facebook Content Server
get FB resource
request FB access
«http redirect»
authorize
permission form
permission form
user permission
process permission
«http redirect»
alt
[permission granted]
FB authorization code
FB authorization code
access token
access FB user protected resource
user protected resource
user protected resource
user protected resource
[no permission]
no authorization
FB resource not available
FB resource not available


:customer
:vending machine
:smart Card Reader
:item
insertCard()
checkCardValidity()
true
chooseItem()
isAvailable()
true
sellItem()
updateCardAmount()
getPrice()
releaseCard()


sd submit_commens
:window
«javascript» :Comments
«servlet» :DWRServlet
«service» :DWRService
«javascript» :PluckRequest Batch
«service» :PluckService
validate()
validate()
«create»
«ajax» :Proxy
«ajax»
«ajax»
validate()
- form fields validation
- CAPTCHA check
errors
errors
«callback»
{10..200ms}
opt
[no errors]
post_comments()
BeginRequest()
«create»
«ajax» :Proxy
«ajax»
postComments()
«json»
{10..100ms}
{1s..4s}
«callback»
ref
Handle errors from Pluck
ref
Request all comments
{10..100ms}

Why not just code it?

- Sequence diagrams can be somewhat close to the code level. So why not just code that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not EVERY line of code is drawn on the diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages)
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)

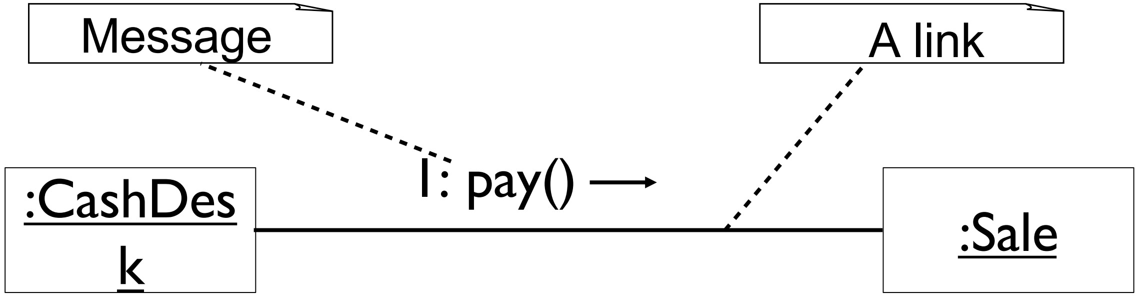
Collaboration/Communication diagram

- A collaboration diagram describes the interaction between objects
  - A collaboration diagram is a graph whose
    - nodes represent object
    - edges represent the communication between objects
  - The temporal ordering of messages is represented by a **numbering** of messages
  - Collaboration diagram is an extension of class diagram

Collaboration diagram

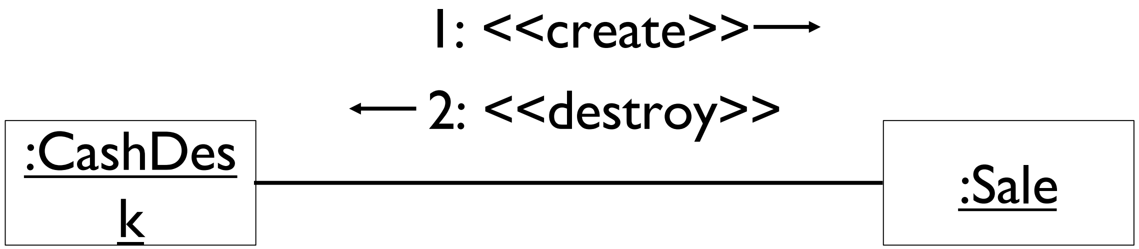
- Links
  - A link shows the sending of a message from an object to another object
  - Formally, a link is an instance of an association

- Messages
  - Each message between objects is presented by an expression of message and an arrow showing the direction of the message



Collaboration diagram

- □ “creation” message and “destruction” message



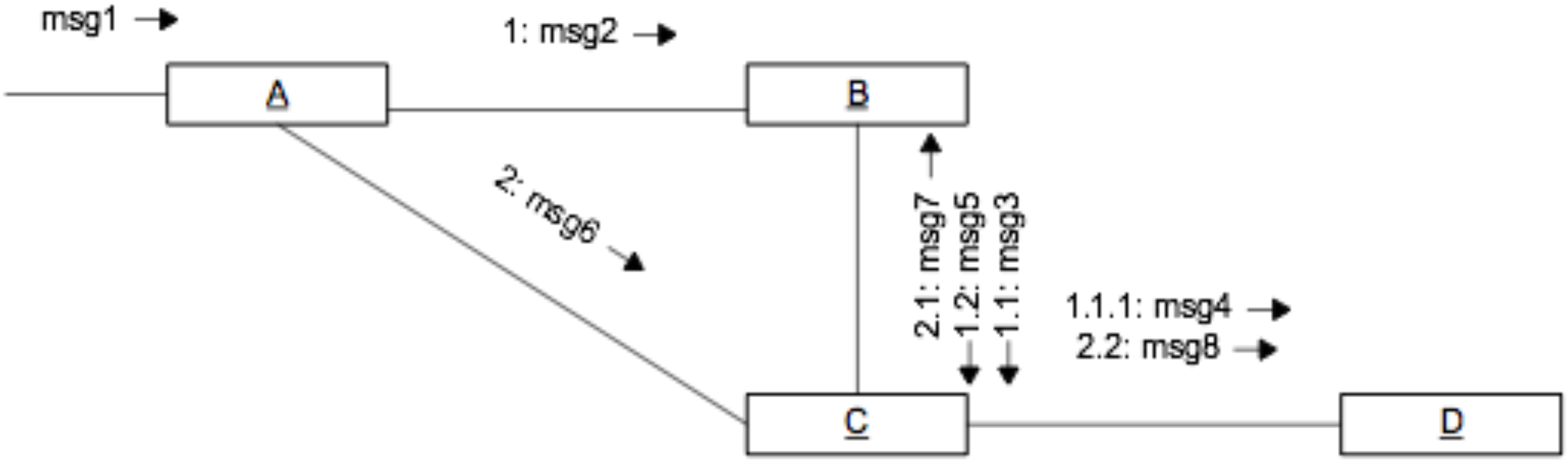
1: <<create>>

2: <<destroy>>

:CashDesk

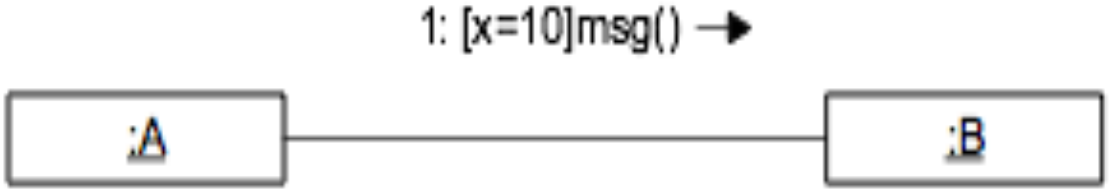
:Sale

- □ Message numbering

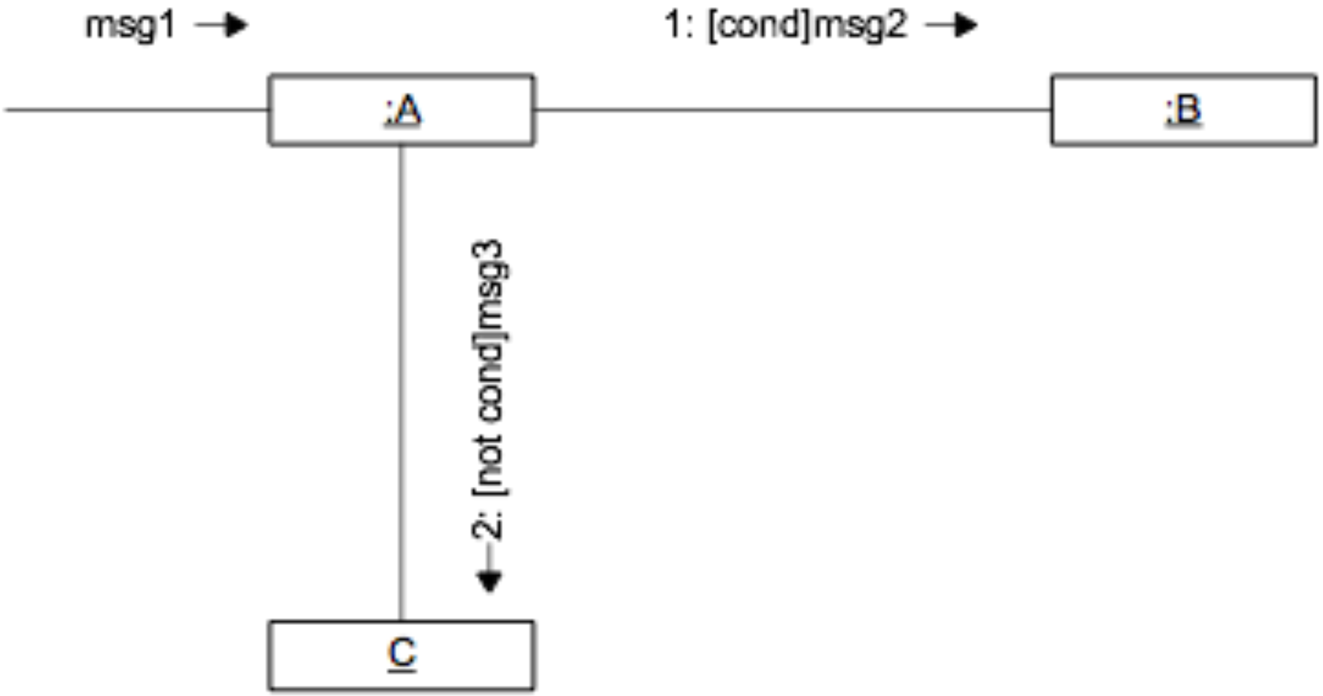


Collaboration diagram

- Conditional message

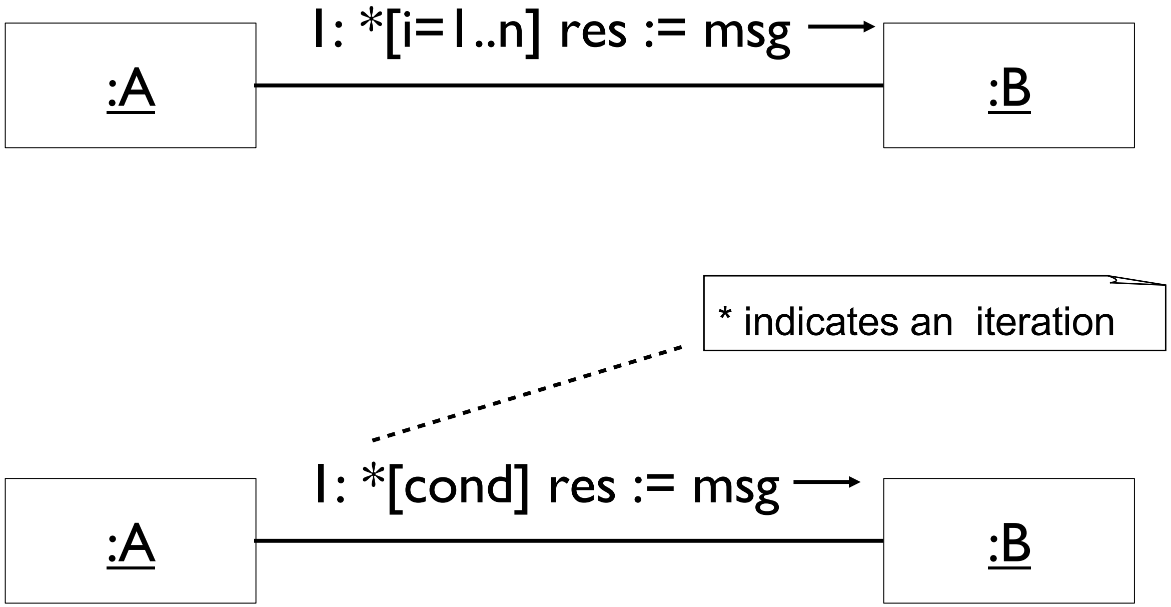


- Modelling a decision



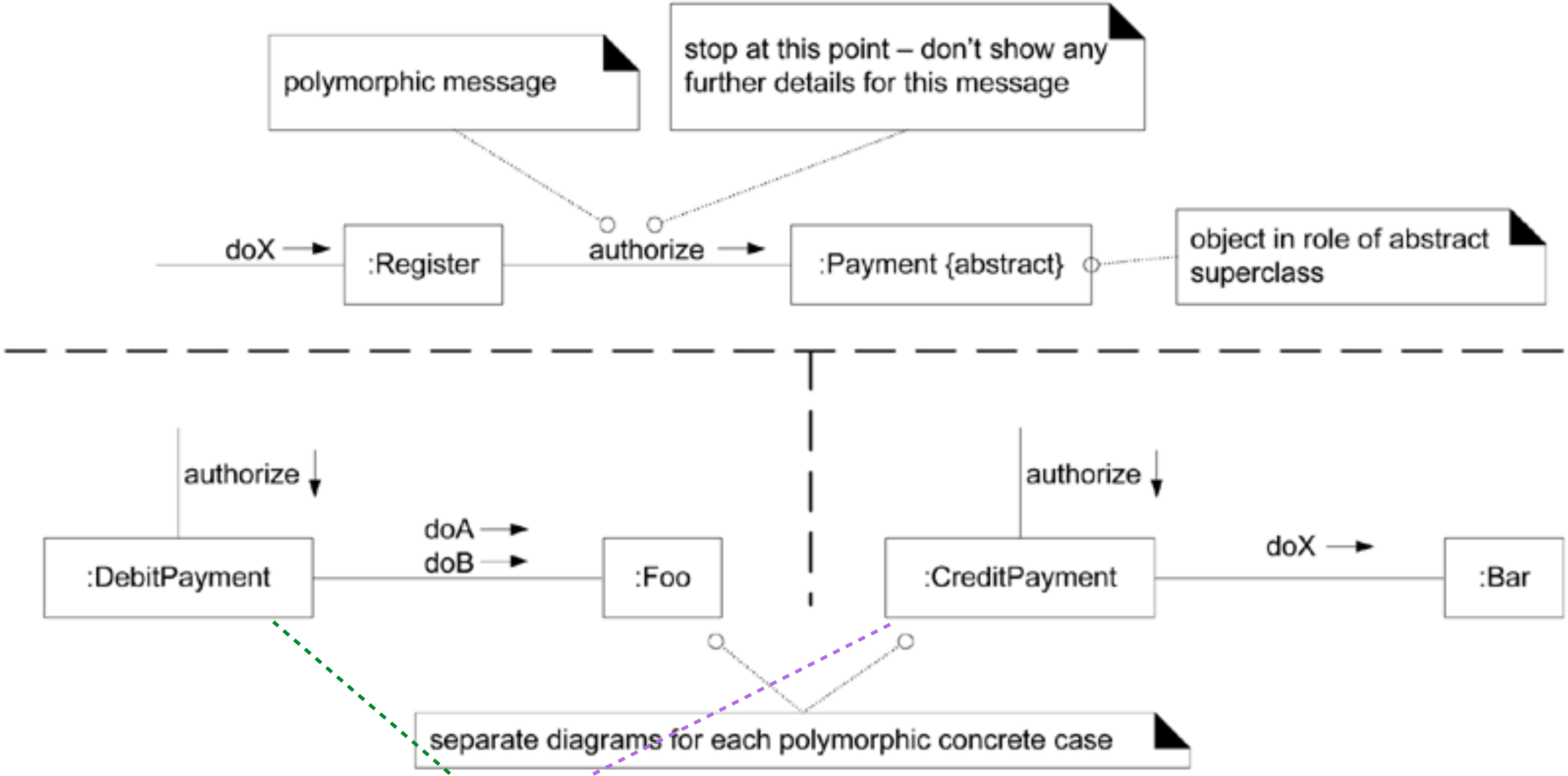
Collaboration diagram

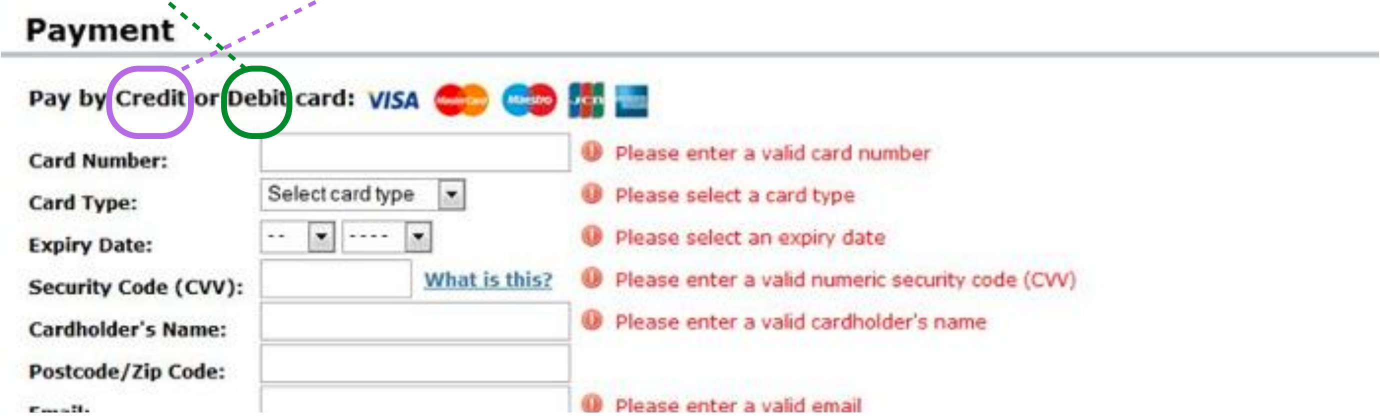
- Modelling an iteration



Collaboration diagram

- Modelling a polymorphic message


polymorphic message
stop at this point – don't show any further details for this message
doX
:Register
authorize
:Payment {abstract}
object in role of abstract superclass
authorize
:DebitPayment
doA
doB
:Foo
authorize
:CreditPayment
doX
:Bar
separate diagrams for each polymorphic concrete case



Payment

Pay by Credit or Debit card:

Card Number: — Please enter a valid card number

Card Type: Select card type — Please select a card type

Expiry Date: -- ---- — Please select an expiry date

Security Code (CVV): What is this? — Please enter a valid numeric security code (CVV)

Cardholder's Name: — Please enter a valid cardholder's name

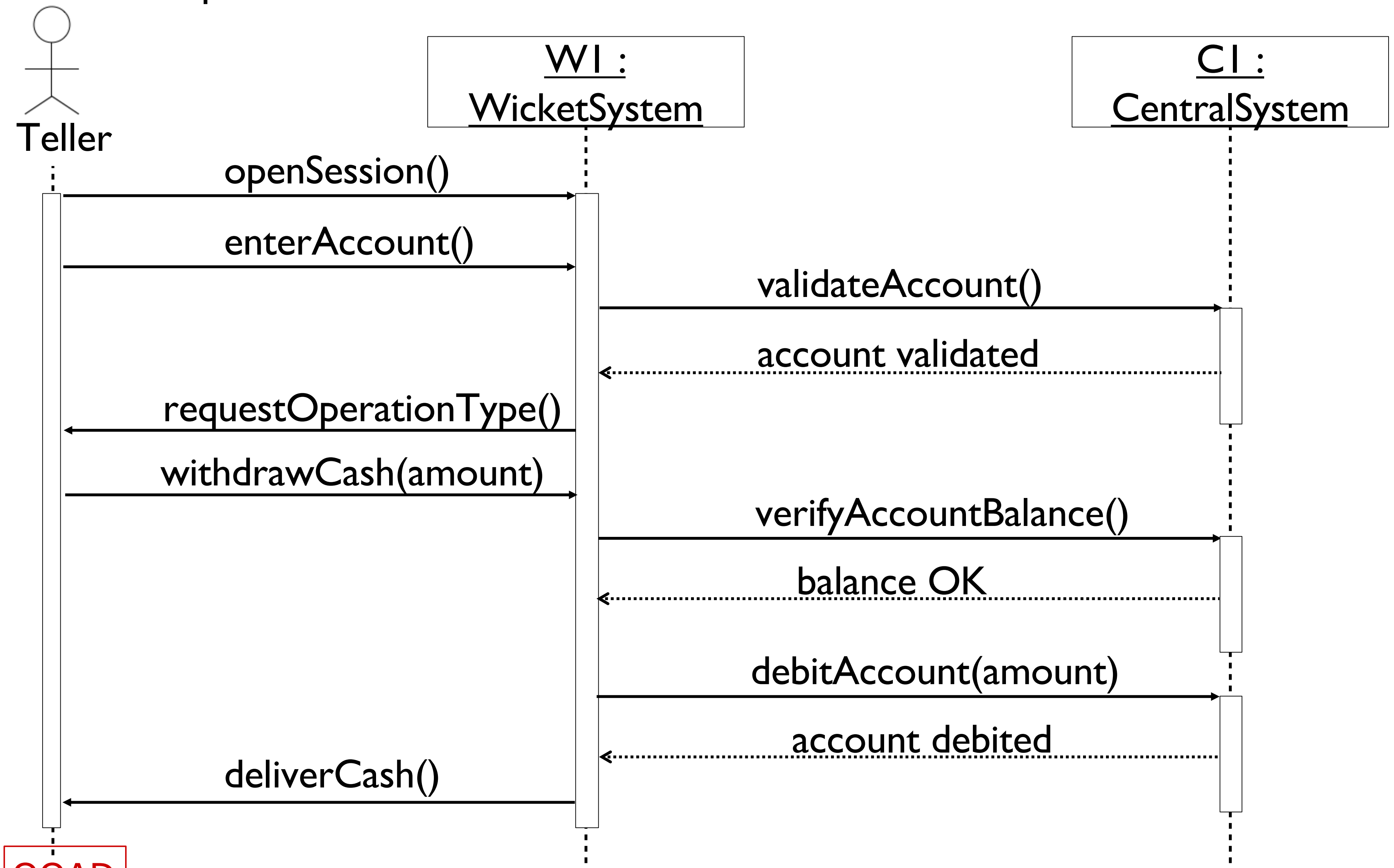
Postcode/Zip Code:

Email: — Please enter a valid email

Cash withdrawal at the bank

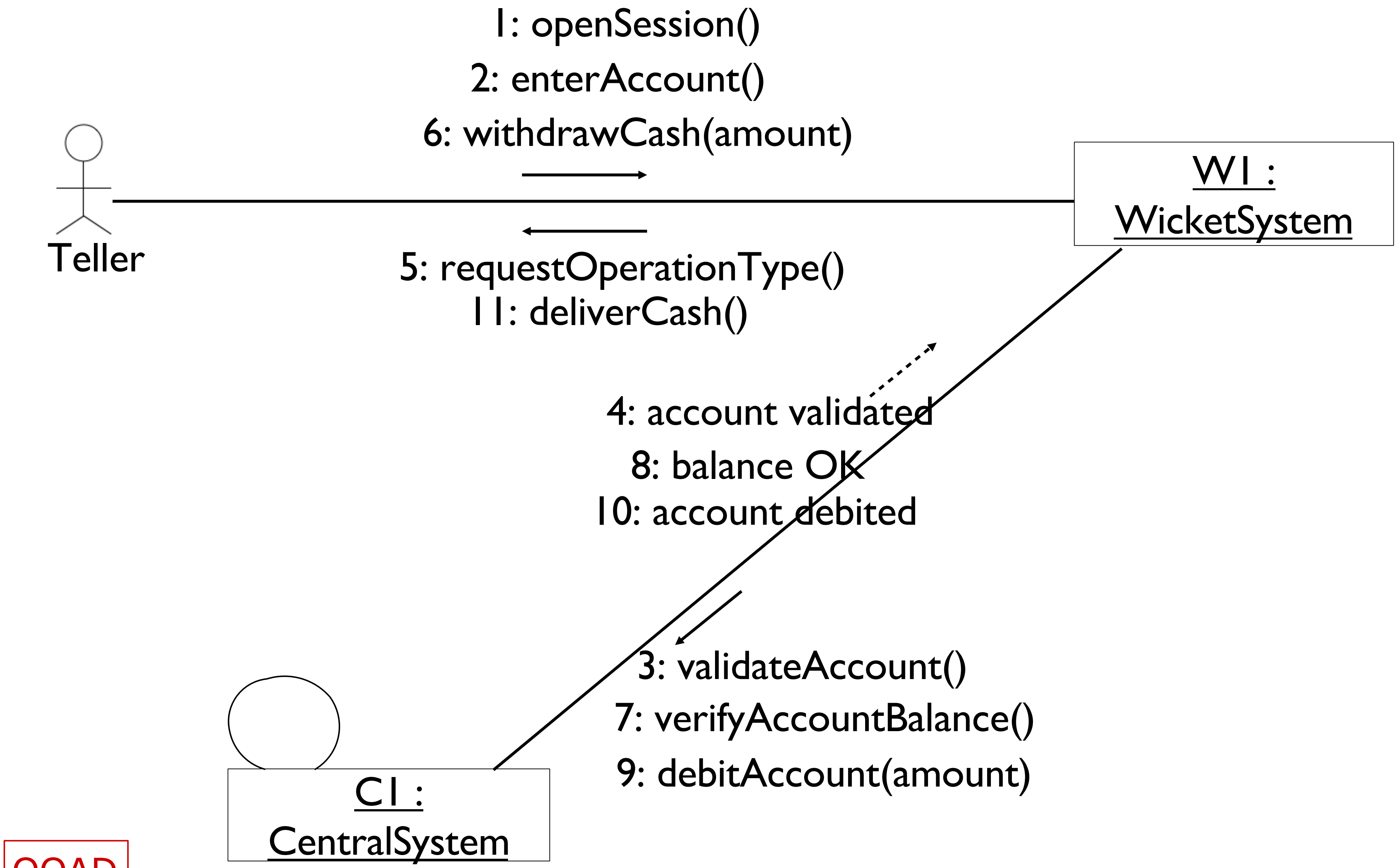
Sequence diagram

- Example: Cash withdrawal at the bank



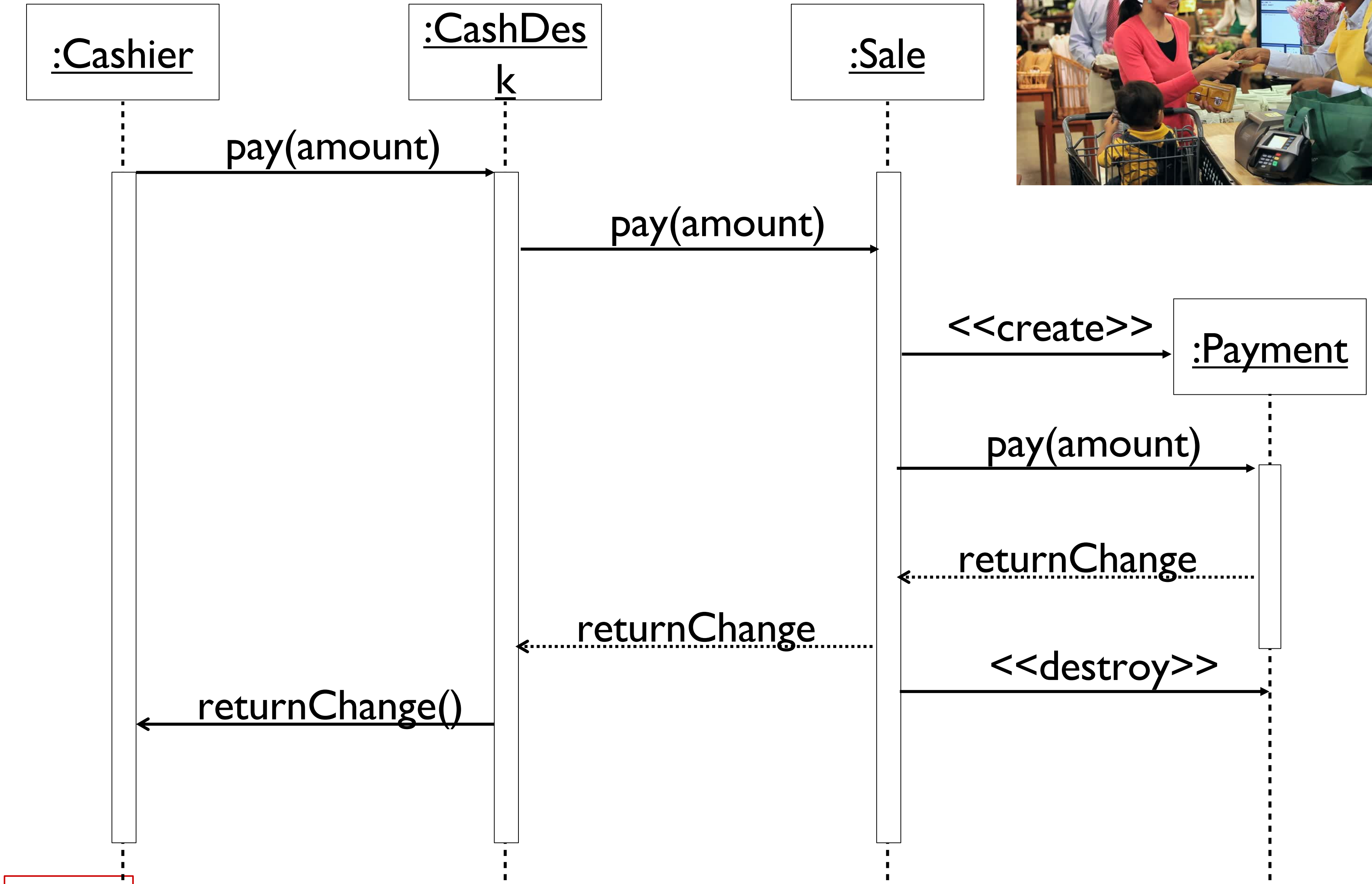
Collaboration diagram

- Example: cash withdrawal in the bank

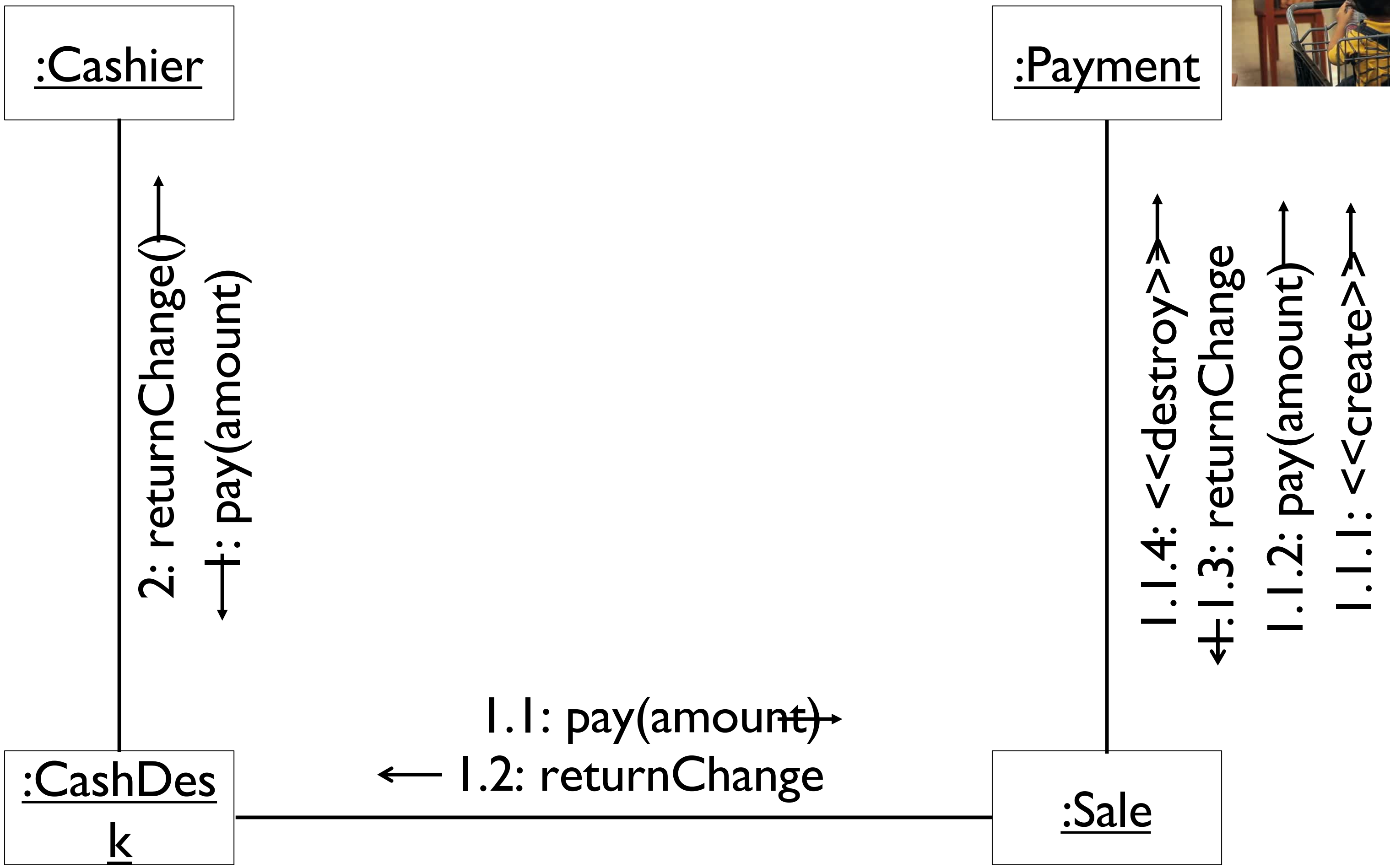


Use-case “cash payment”

Sequence diagram



Collaboration diagram

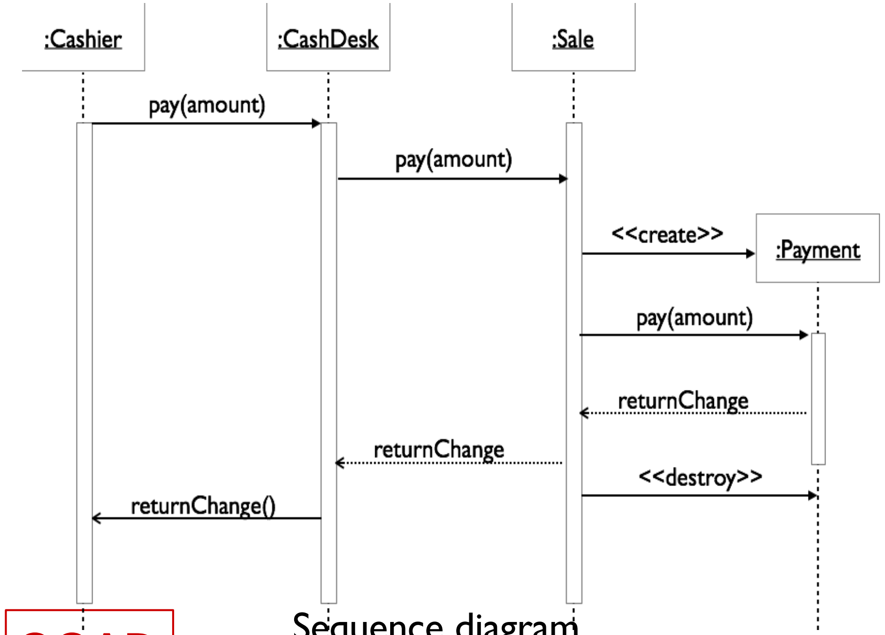


Sequence diagram v.s. Collaboration diagram

- Both sequence diagram and collaboration diagram are alternate representations of an interaction
- Sequence diagram
  - is a graphical view of a scenario
  - shows object interaction in a time-based sequence of what happens first, what happens next
  - establishes the roles of objects and help provide essential information to determine class responsibilities and interfaces
  - is normally associated with a use-case
- Collaboration diagram
  - shows how object associate with each other (objects, links and messages)
  - provides the structural relationships between objects

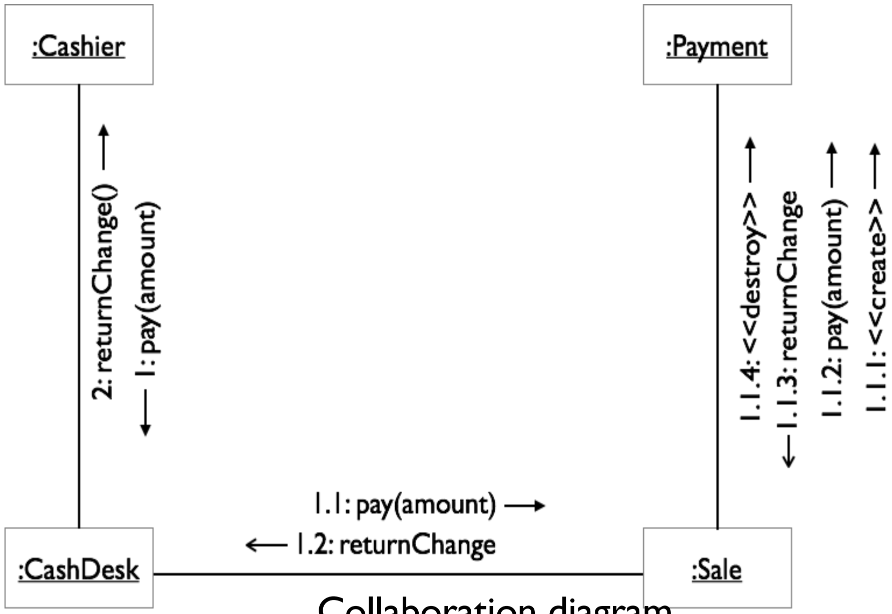
Sequence diagram v.s. Collaboration diagram

- Sequence diagram
  - 👍 Clearly shows the temporal ordering of messages
  - 👎 Consumes space
- Collaboration diagram
  - 👍 Is preferable when the interaction is deduced from the class diagram
  - 👍 Consumes less space
  - 👎 Is difficult to see the sequence of messages



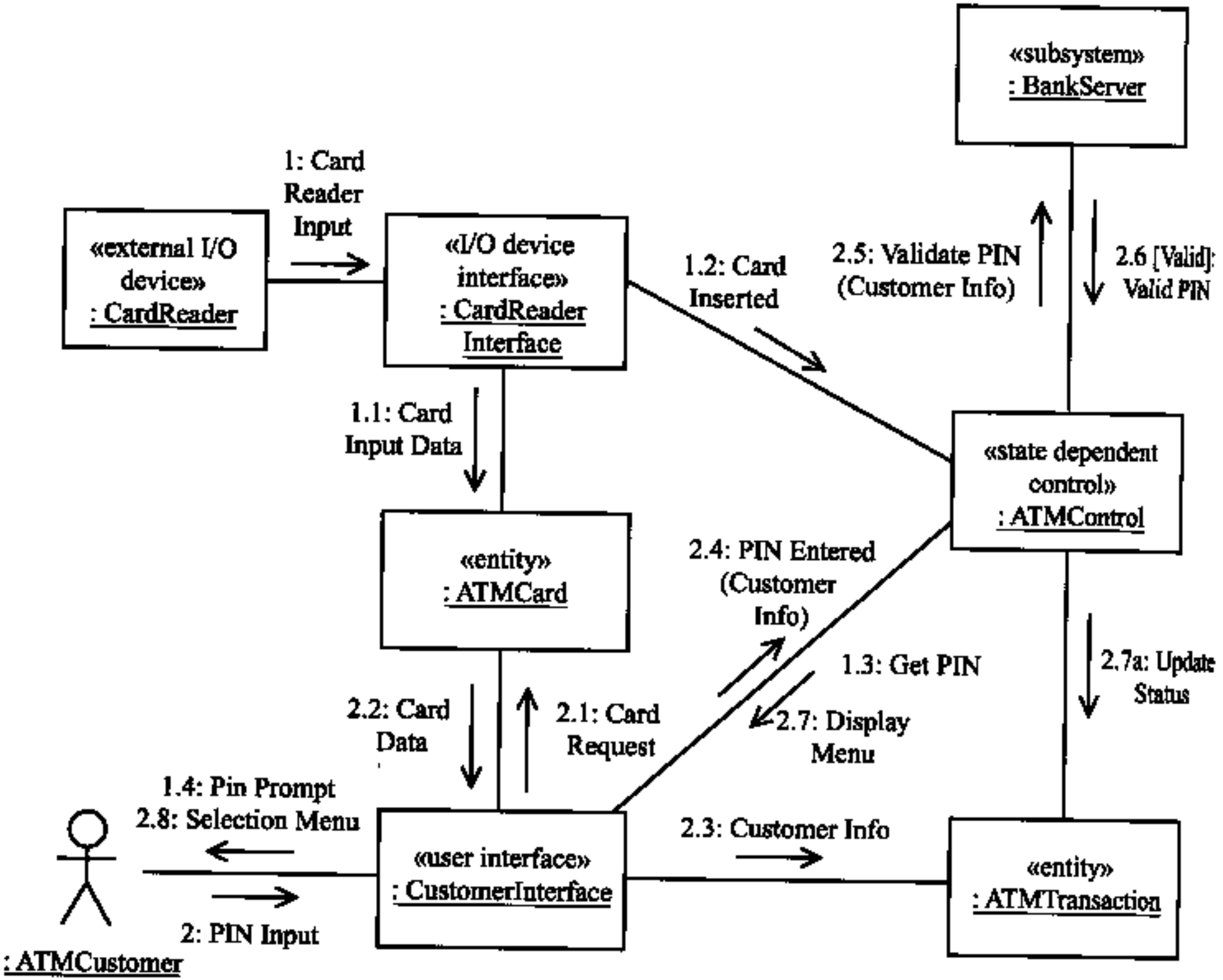
Sequence diagram

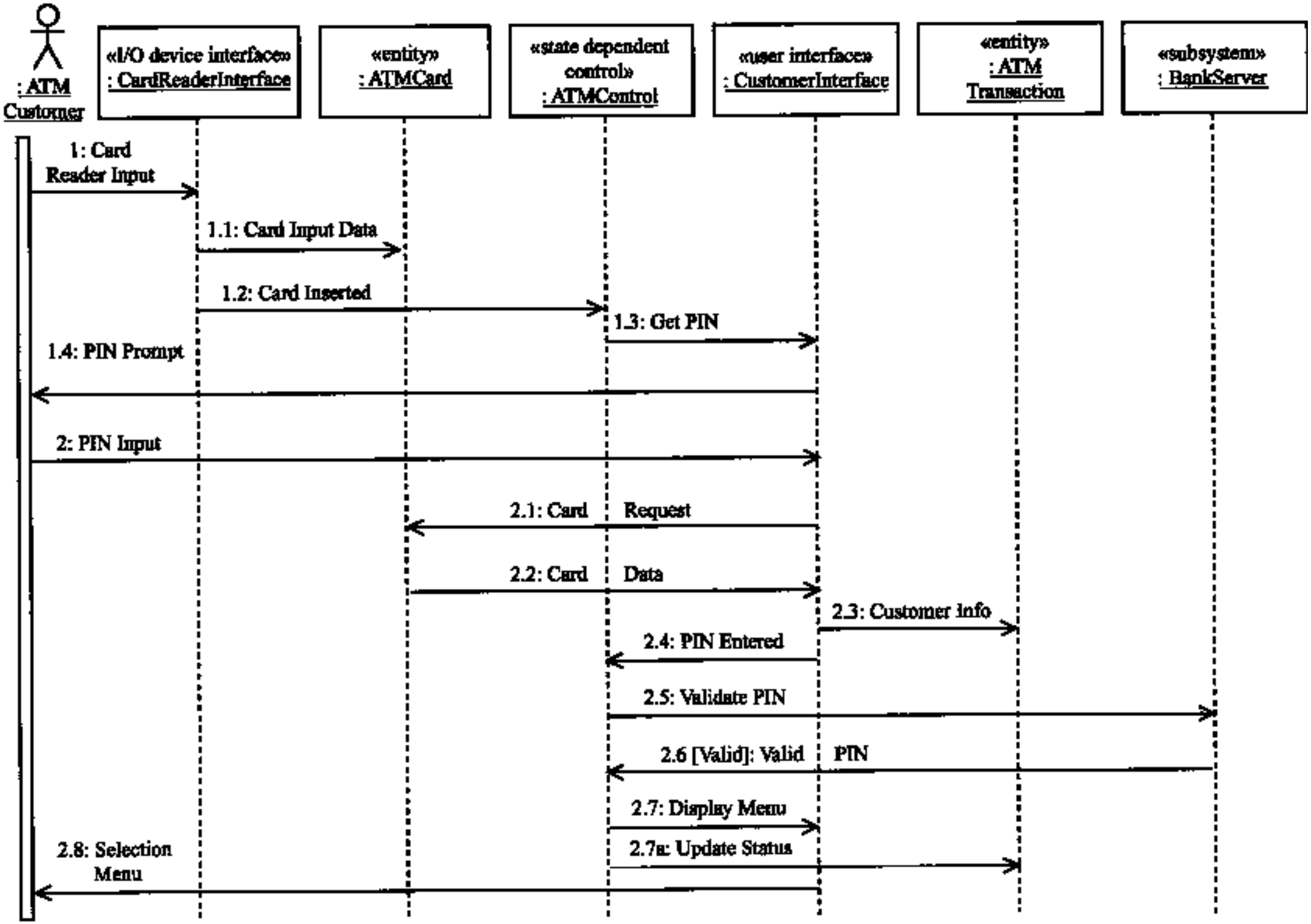
v.s



Collaboration diagram

- Let's do a sequence diagram for the following casual use case, *Add Calendar Appointment* :
  - 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.


«subsystem»
: BankServer
1: Card Reader Input
«external I/O device»
: CardReader
«I/O device interface»
: CardReader Interface
1.2: Card Inserted
2.5: Validate PIN (Customer Info)
2.6 [Valid]: Valid PIN
1.1: Card Input Data
«state dependent control»
: ATMControl
«entity»
: ATMCard
2.4: PIN Entered (Customer Info)
1.3: Get PIN
2.7a: Update Status
2.2: Card Data
2.1: Card Request
2.7: Display Menu
1.4: Pin Prompt
2.8: Selection Menu
«user interface»
: CustomerInterface
2.3: Customer Info
«entity»
: ATMTransaction
2: PIN Input
: ATMCustomer


: ATM Customer
«I/O device interface» : CardReaderInterface
«entity» : ATMCard
«state dependent control» : ATMControl
«user interface» : CustomerInterface
«entity» : ATM Transaction
«subsystem» : BankServer
1: Card Reader Input
1.1: Card Input Data
1.2: Card Inserted
1.3: Get PIN
1.4: PIN Prompt
2: PIN Input
2.1: Card Request
2.2: Card Data
2.3: Customer info
2.4: PIN Entered
2.5: Validate PIN
2.6 [Valid]: Valid PIN
2.7: Display Menu
2.7a: Update Status
2.8: Selection Menu

Fun example

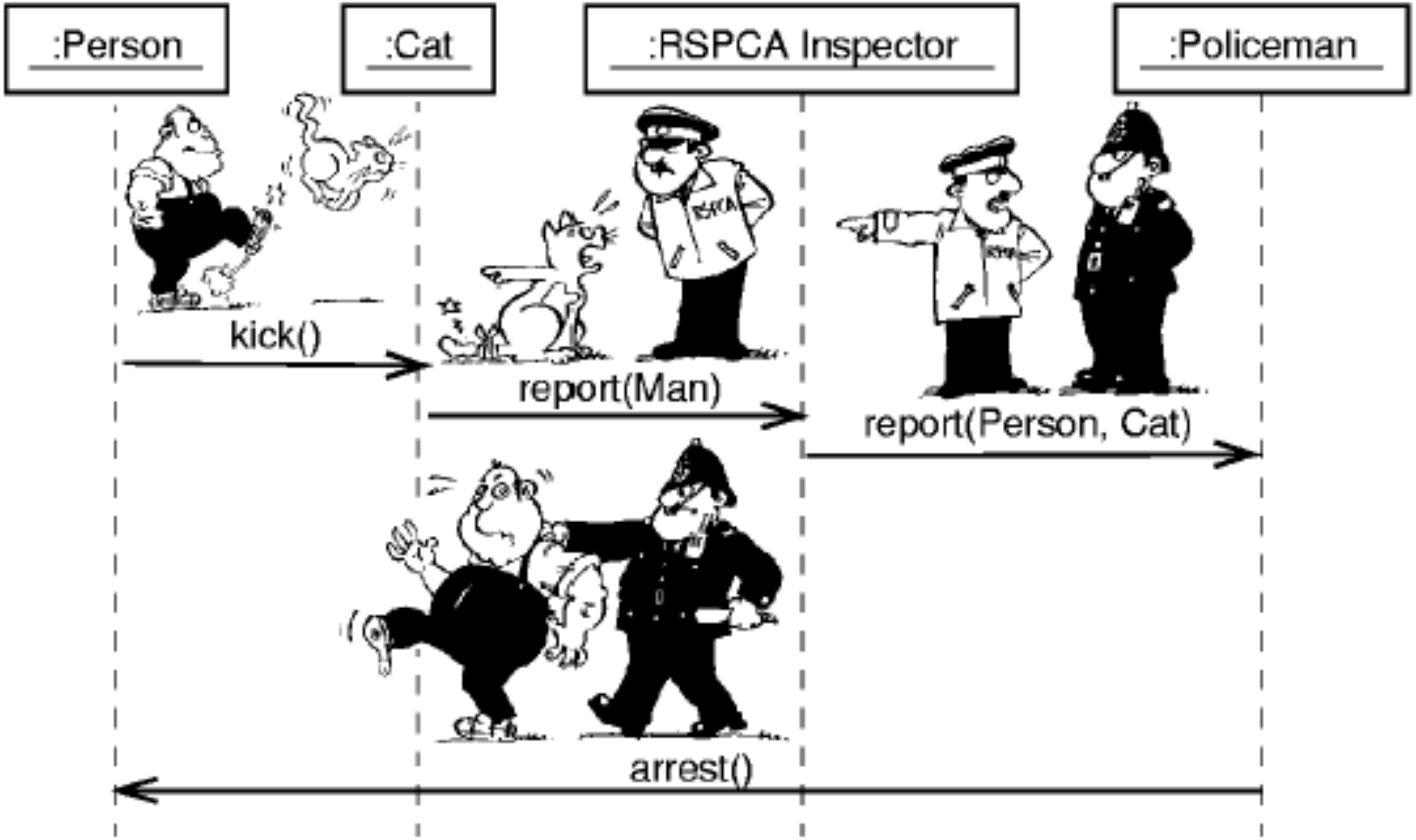
:Cat

:Policeman

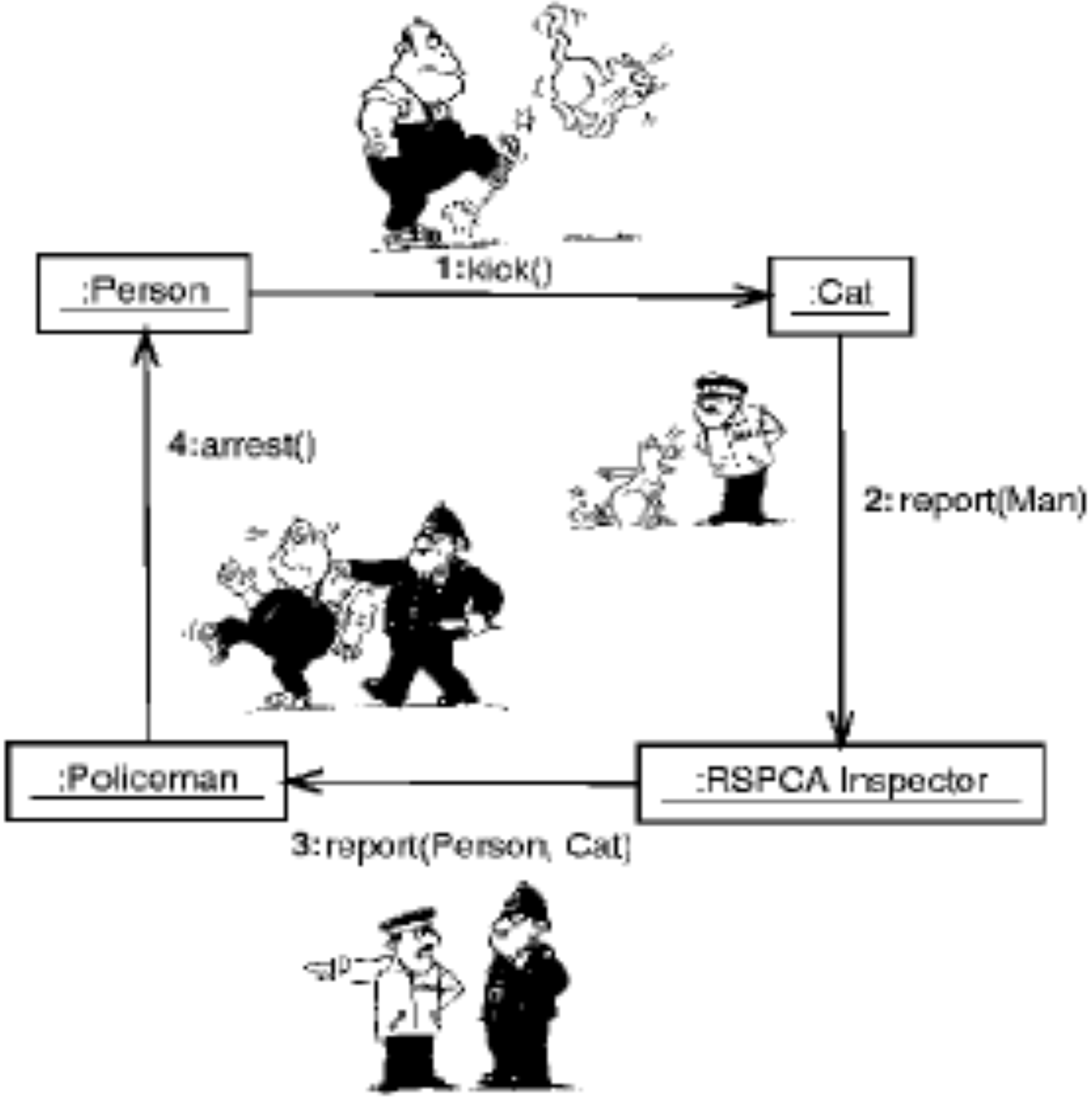
:Person

:RSPCA Inspector

Fun example: Sequence diagram



Fun example: Collaboration diagram


:Person
1:kick()
:Cat
2:report(Man)
4:arrest()
:Policeman
:RSPCA Inspector
3:report(Person, Cat)