SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Behavioral Diagrams

Interaction Diagram

building blocks of UML - Diagrams

Graphical representation of a set of elements.

Represented by a connected graph: Vertices are things; Arcs are relationships/behaviors.

5 most common views built from

UML 1.x: 9 diagram types.

Structural Diagrams

Represent the static aspects of a system.

- Class;
  Object
- Component
- Deployment

Behavioral Diagrams

Represent the dynamic aspects.

- Use case
- Sequence;
  Collaboration
- Statechart
- Activity

UML 2.0: 12 diagram types

Structural Diagrams

- Class;
  Object
- Component
- Deployment
- Composite Structure
- Package

Behavioral Diagrams

- Use case
- Statechart
- Activity

Interaction Diagrams

- Sequence;
  Communication
- Interaction Overvie
- Timing

Sequence Diagrams

Behavioral Diagrams

- Use case
- Statechart
- Activity

Interaction Diagrams

- **Sequence**; ***Communication***
- Interaction Overview
- Timing

Interaction Diagrams (seq and coll)

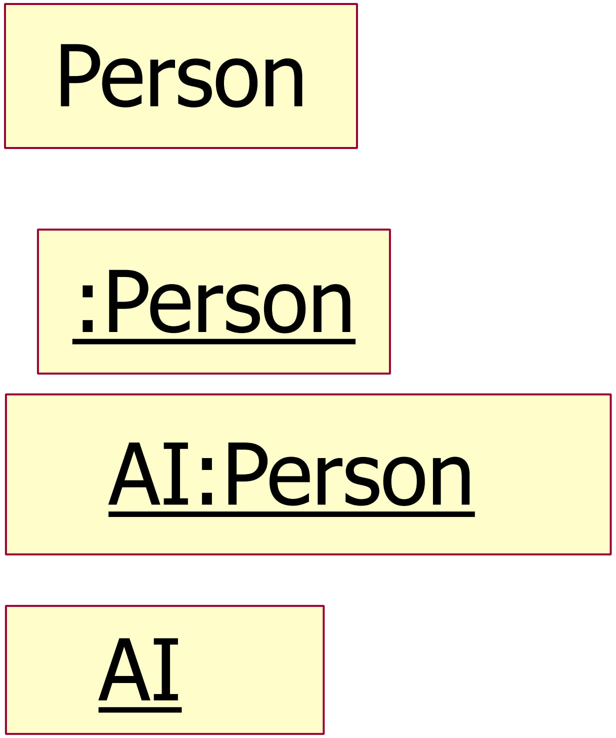
- show the interaction of *any kind of instance (classes, interfaces, components, nodes and subsystems);*
- messages sent/received by those objects/instances (invocation, construction/destruction, of an operation)
- realizes use cases to model a scenario
- Interaction types (these are isomorphic, when no loops or branching)
  - Sequence diagram —emphasizes the time ordering of messages.
  - Communication (Collaboration) diagram — emphasizes the structural organization of objects that directly send and receive messages.

- Objects within an interaction can be:
  - Concrete: something from the real world. (e.g., **John: Person**)
  - Prototypical: representative instance of something from the real world (e.g., **p: Person**)
    - Communication diagrams use (strictly) prototypical things.
    - Prototypical instances of interfaces and abstract types are valid.

Interaction diagrams: notation

- The following notation is used in the UML for classes and objects:
  - Class
  - Instance of a Class

    (object without a name)
  - Named instance of a class (named object)
  - Named object only (shown without class)



Person

:Person

AI:Person

AI

Sequence diagrams

- A sequence diagram shows relations between objects.
- It should be read from left to right and from top to bottom.
- At the top of the diagram are **names of objects that interact** with each other.

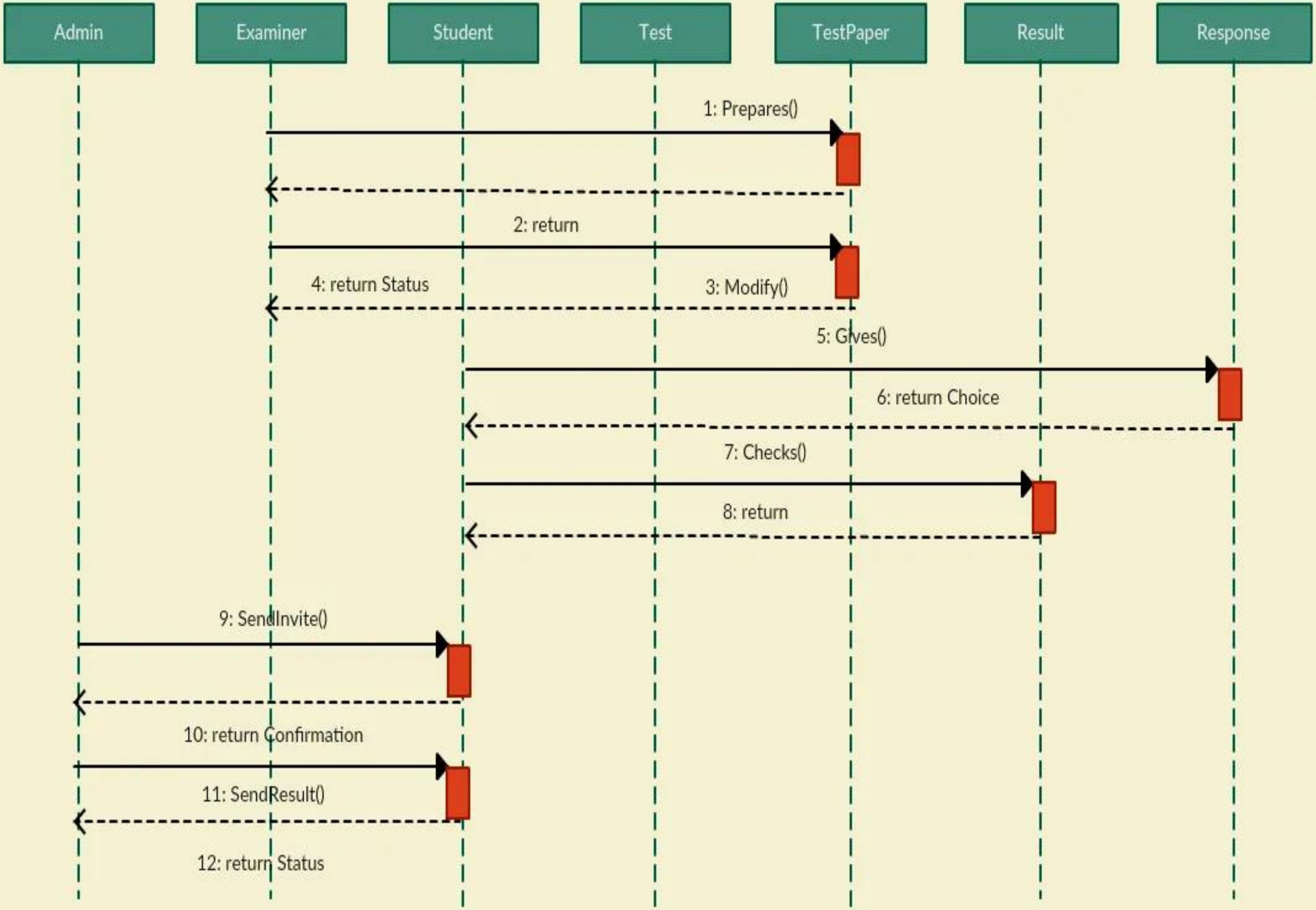
  These are the concepts in the **conceptual model**.
- When the course of events is initiated by an actor the actor symbol is displayed as the leftmost object.
- A sequence diagram consists of a **group of objects** that are represented by lifelines, and the messages that they exchange over time during the interaction.
- Sequence diagrams can also show the **control structures between objects.**

Steps to be for Sequence Diagram

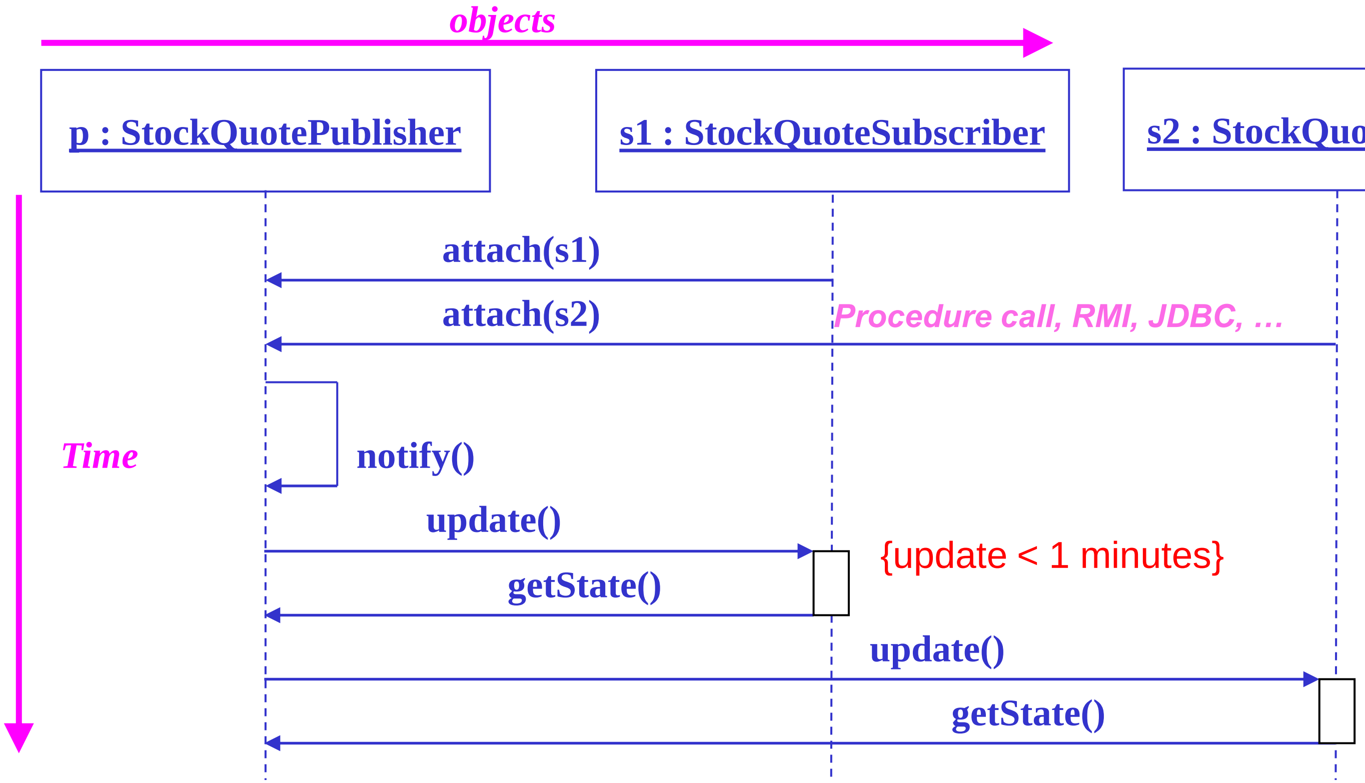
- Identify the Scenario:
- List the Participants:
- Define Lifelines:
- Arrange Lifelines:
- Add Activation Bars:
- Draw Messages:
- Include Return Messages:
- Indicate Timing and Order:

Sequence Diagram Components

- Life lines
- Messages – Create , Delete, Self, Reply, Found, Lost, Guard
- Time
- Delete / Destroy
- Self Loop
- Iteration
- Recursive


Admin
Examiner
Student
Test
TestPaper
Result
Response
1: Prepares()
2: return
4: return Status
3: Modify()
5: Gives()
6: return Choice
7: Checks()
8: return
9: SendInvite()
10: return Confirmation
11: SendResult()
12: return Status

Interaction Diagram: sequence vs communication



object role:ClassName

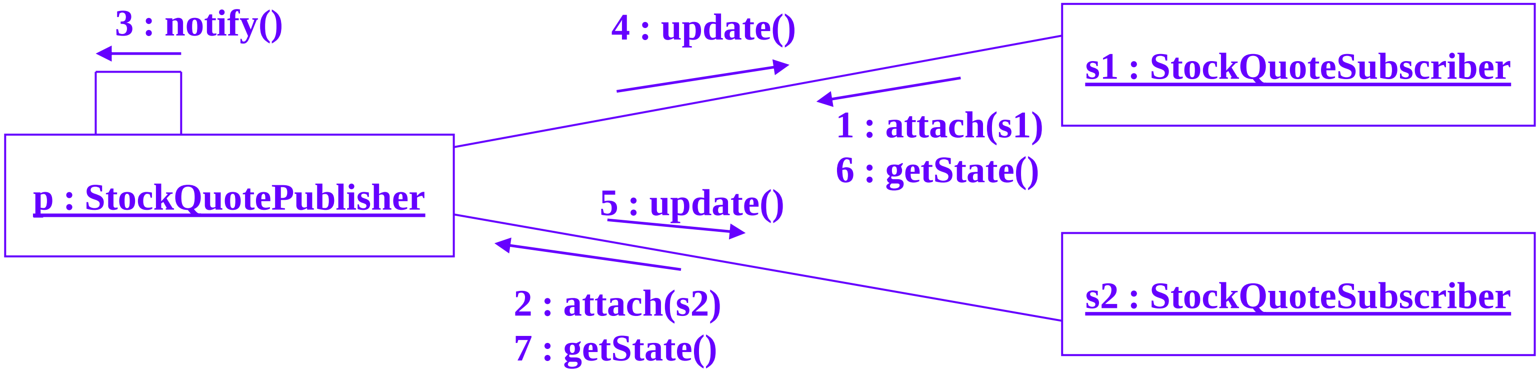
classifiers or their i

use cases or actors

Observer design pattern

Activations (See pg 14)

- Show duration of execution
- Shows call stack
- Return message
  - Implicit at end of activation
  - Explicit with a dashed arrow



Message Guidelines

- Justify Message Names Beside the Arrowhead
- Create Objects Directly
- Apply Operation Signatures for Software Messages
- Use Description (prose) for Messages Involving Human and Organization Actors
- Prefer Names Over Types for Parameters
- Indicate Types as Parameter Placeholders
- Messages to Classes are Implemented as Static Operations
- Apply the <> Stereotype for Use Case Invocations

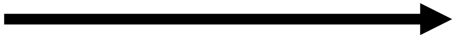
Classifiers

constraints, can be derived, can be stereotyped, tagged value

- Name Objects When You Refer To Them In Messages
- Name Objects When Several of the Same Type Exist
- Apply Textual Stereotypes Consistently
- Apply Visual Stereotypes Sparingly
- Focus on Critical Interactions

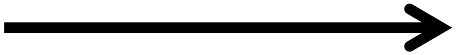
Message Flow Notation

Same as for sequence diagrams

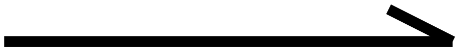


Synchronous

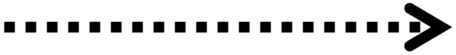
(the sender waits until the responder finishes)



(Flat) Flow of control (the sender doesn't wait for anything from the responder and finishes its' activity; the control is passed to the responder)



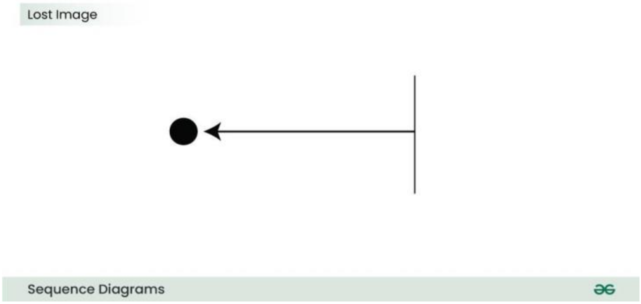
Asynchronous the sender doesn't wait for anything from the responder, but it continues its' own activity



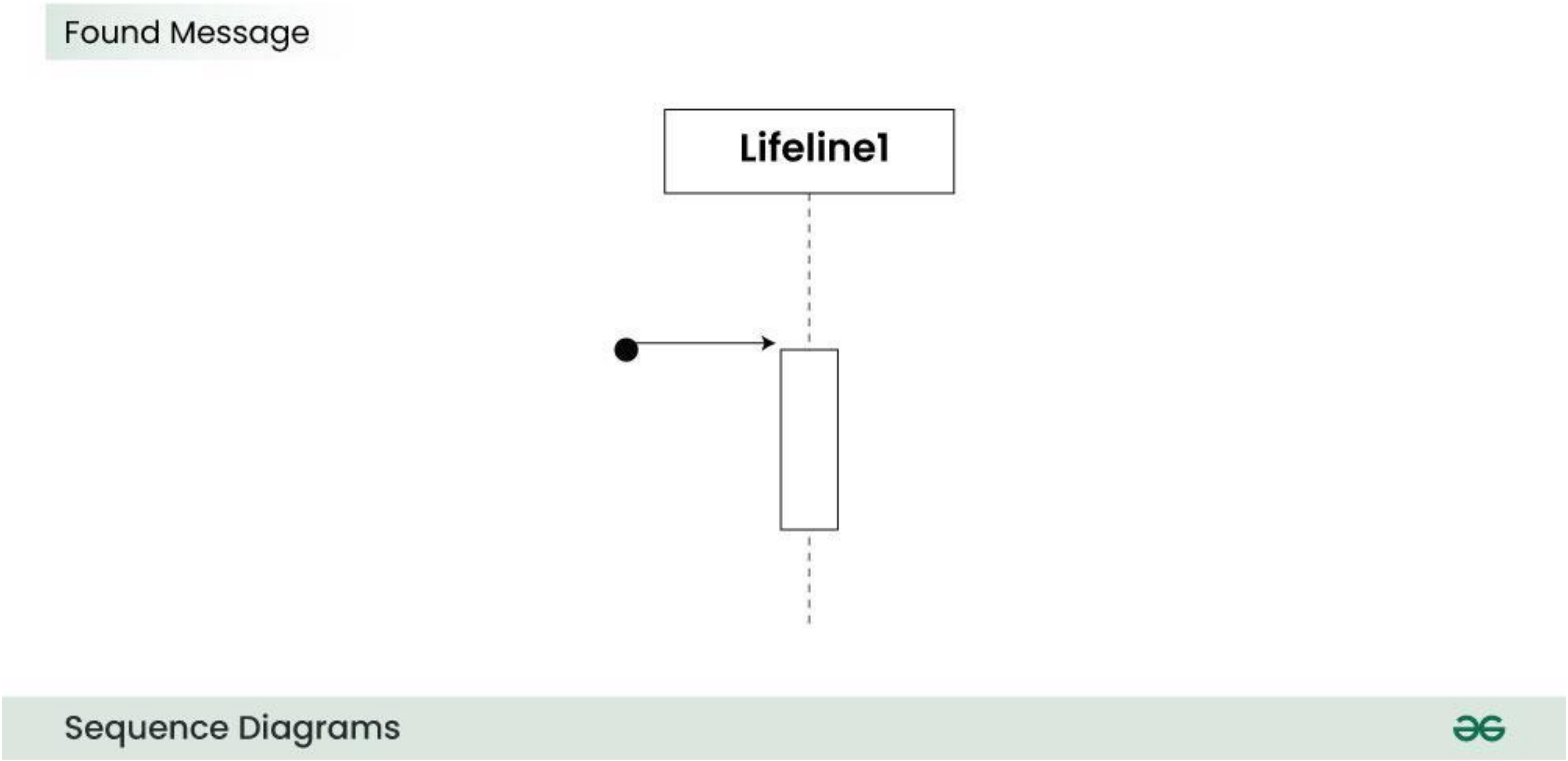
Return

Types of message flows

- A Lost message is used to represent a scenario where the recipient is not known to the system.
- It is represented using an arrow directed towards an end point from a lifeline.



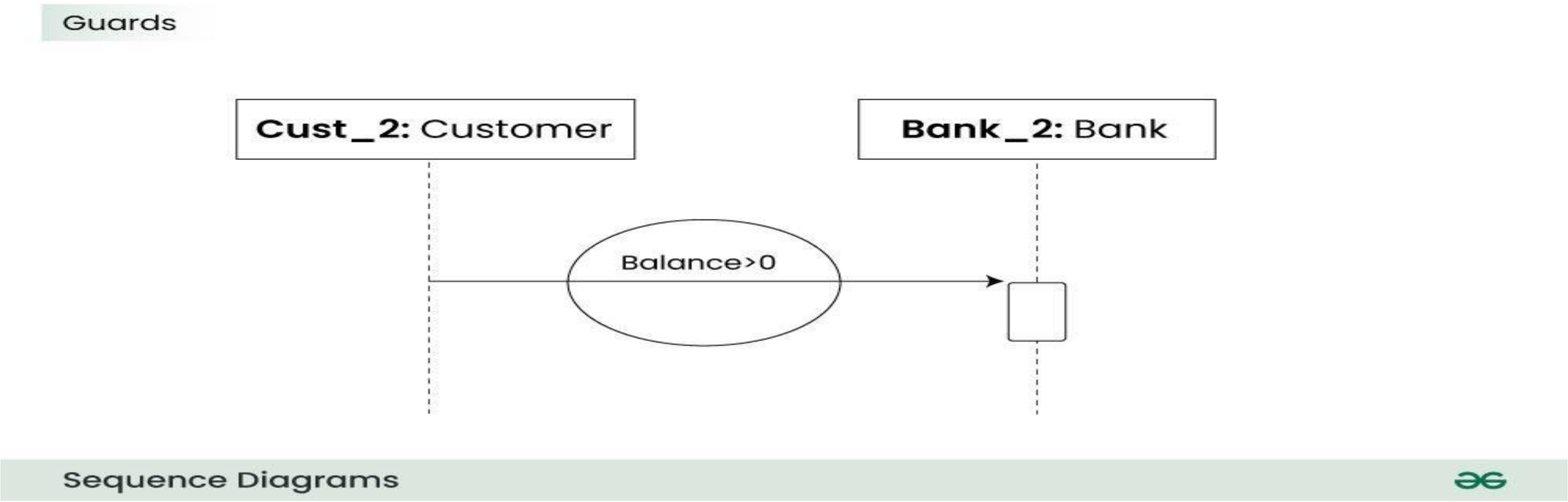
Found Message



Guards play an important role in letting software developers know the **constraints attached** to a system or a particular process.

For example:

In order to be able to withdraw cash, having a balance greater than **zero is a condition** that must be met as shown below.



Iterating Messages

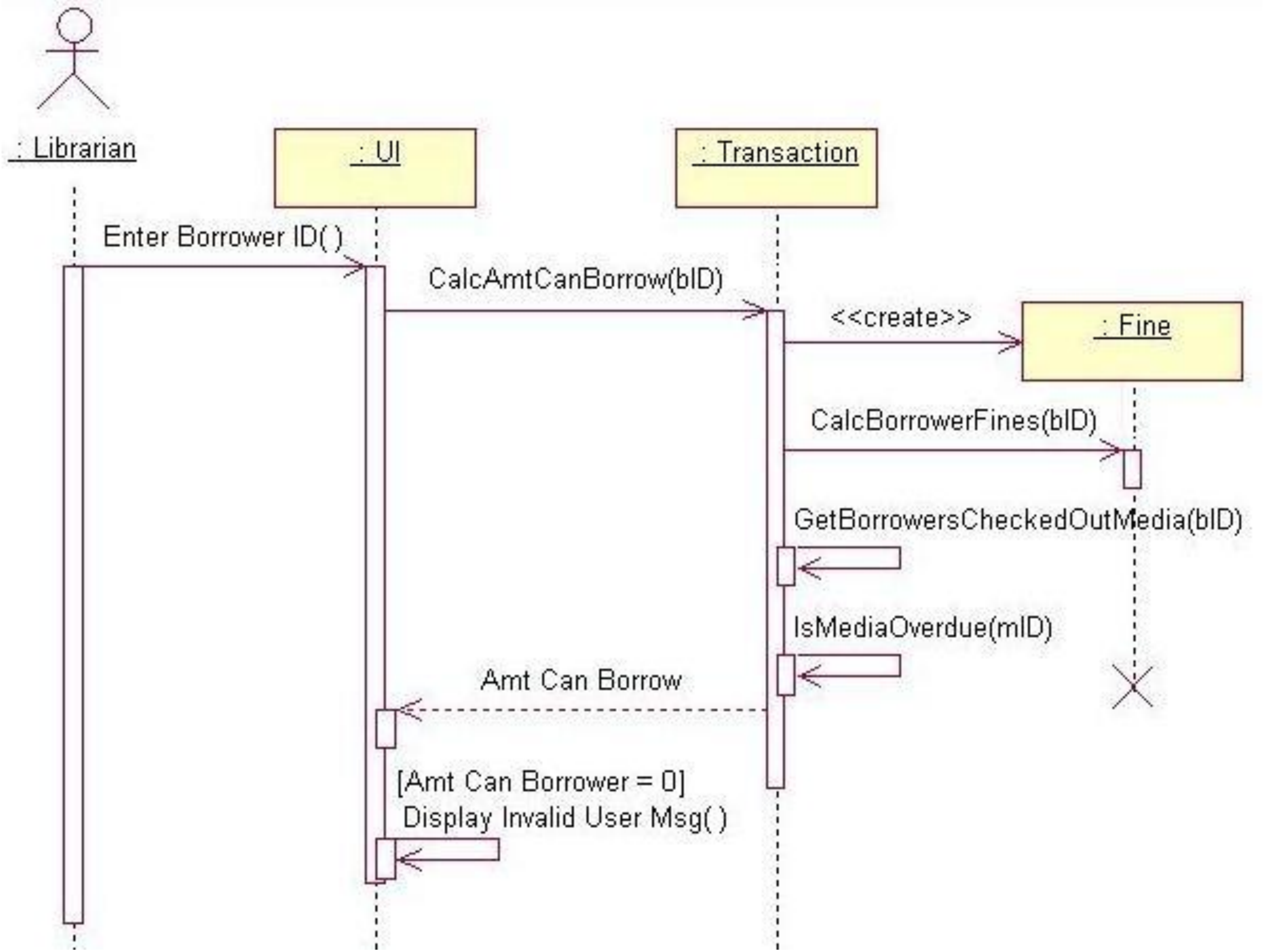
- Collaboration diagrams use syntax similar to sequence diagrams to indicate that either a message iterates (is run multiple times) or is run conditionally
  - – An asterisk (*) indicates that a message **runs more than once**
  - – Or the number of times a message is repeated can be shown by numbers (for example, 1..5)

Conditional Messages

- To indicate that a message is run conditionally, prefix the message sequence number with a conditional [**guard**] clause in brackets

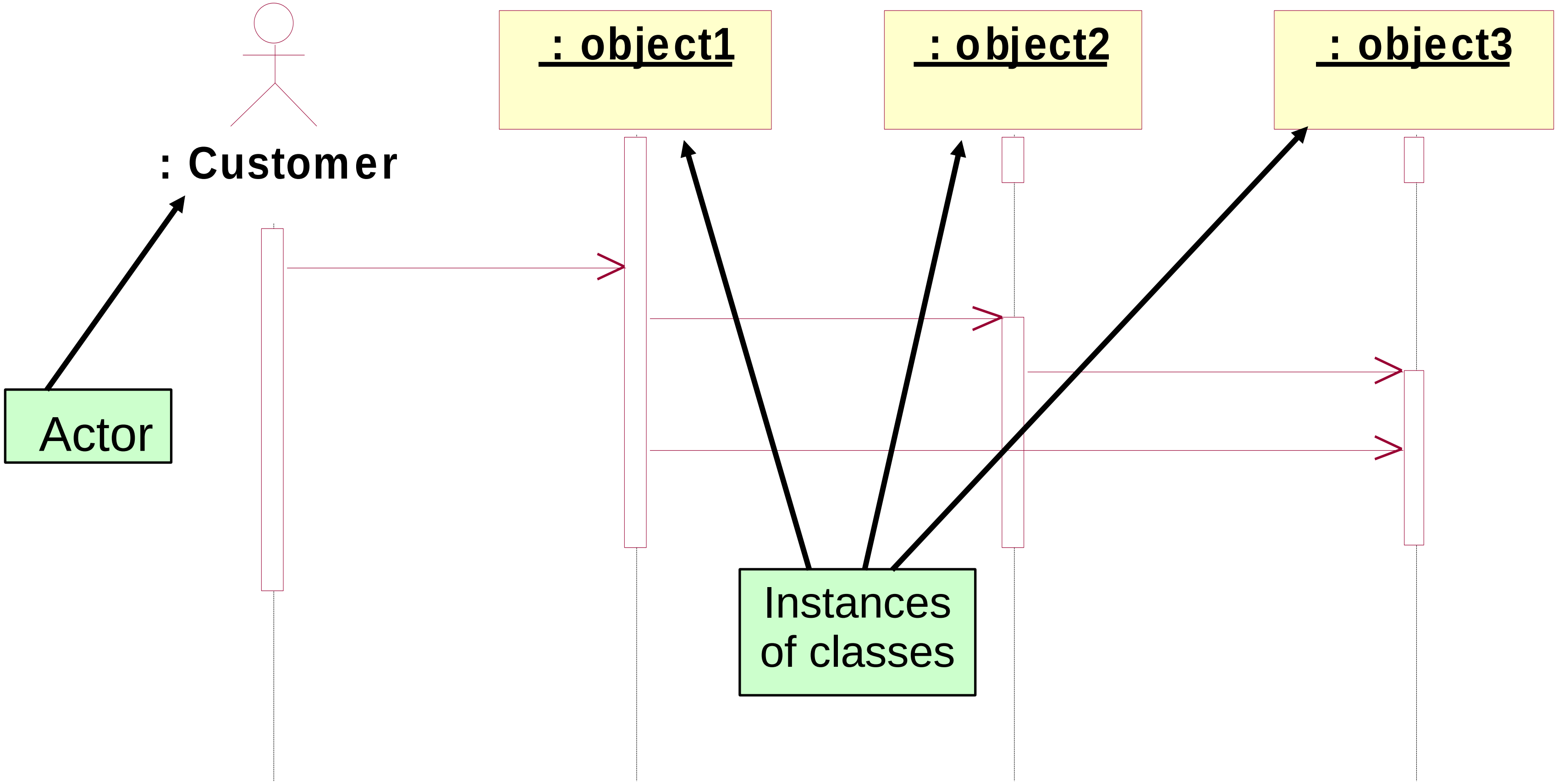
  [x = true]: **[IsMediaOverdue]**

- This indicates that the message is sent only if the **condition is met**

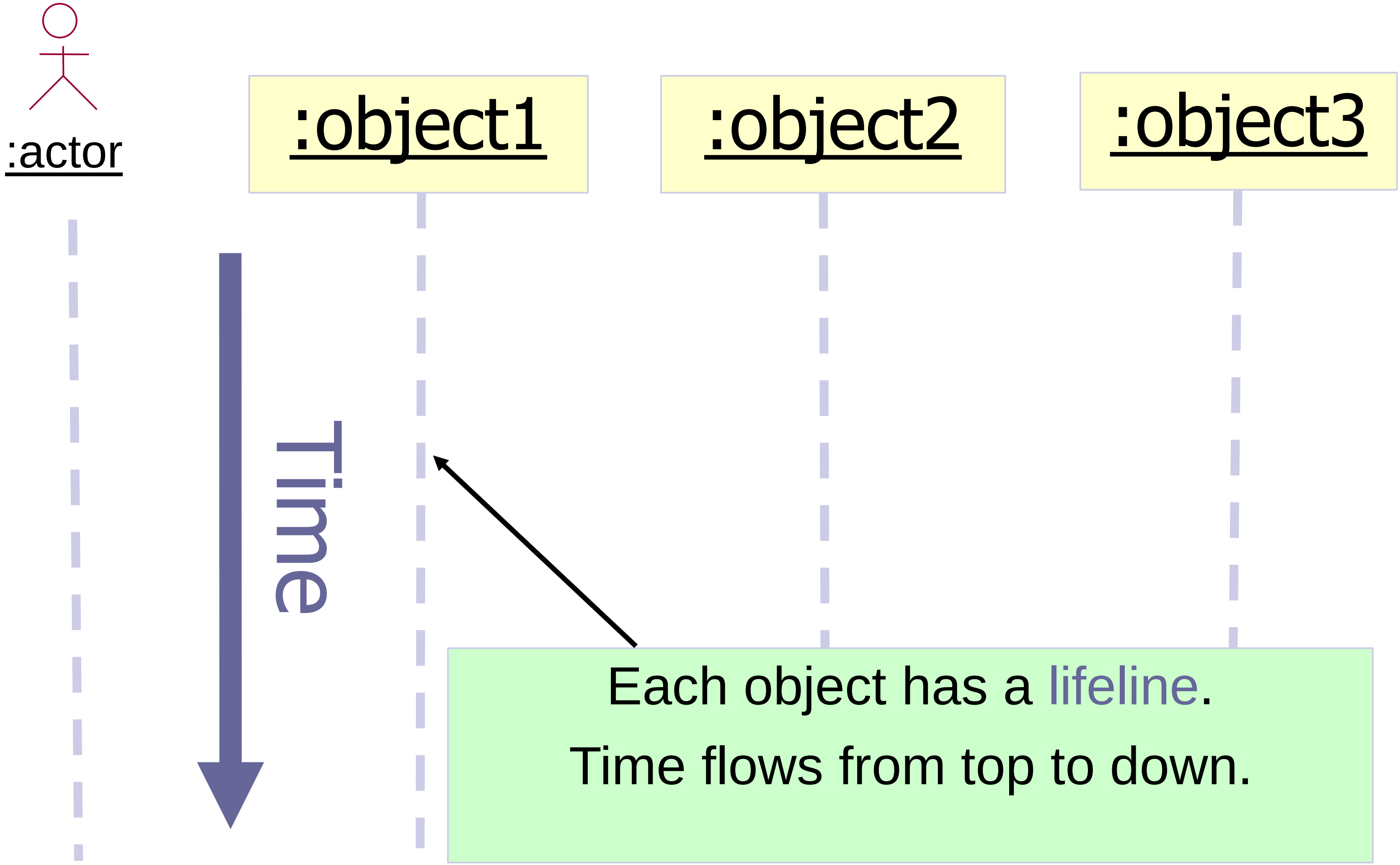


Sequence diagram is better at ‘time ordering’

Sequence diagrams: an example



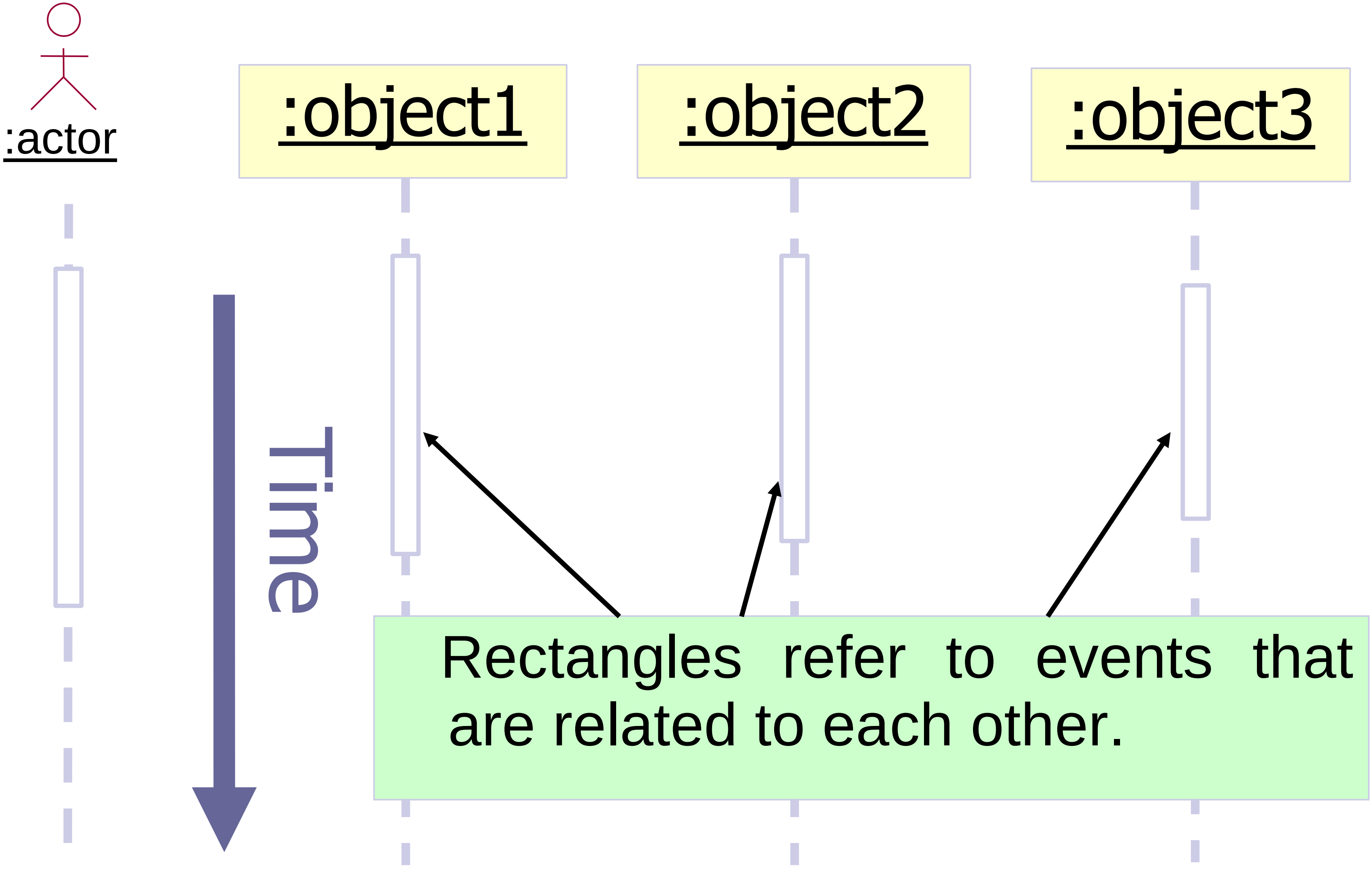
Sequence Diagram: Lifelines



Each object has a lifeline.

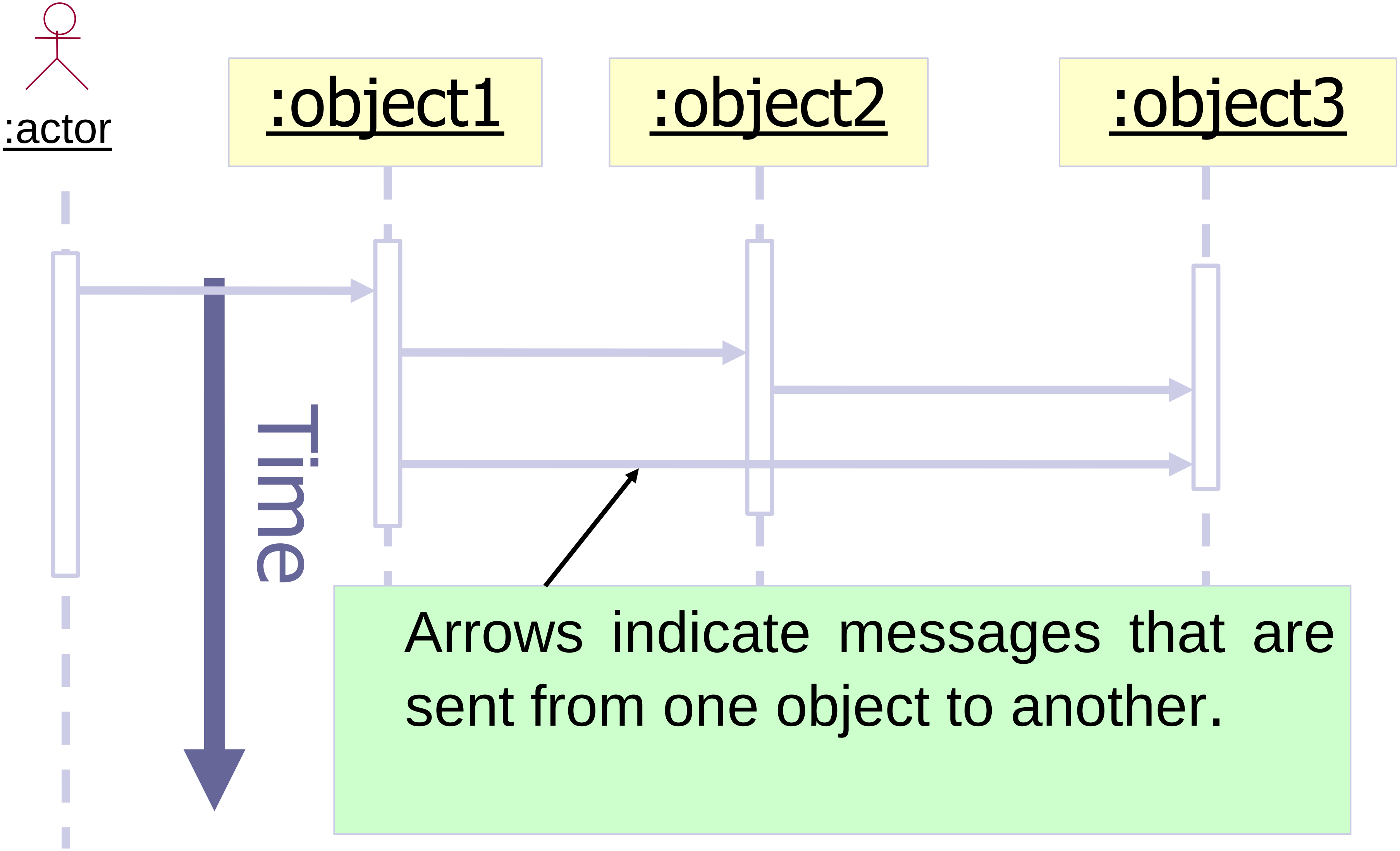
Time flows from top to down.

Sequence diagrams: rectangles



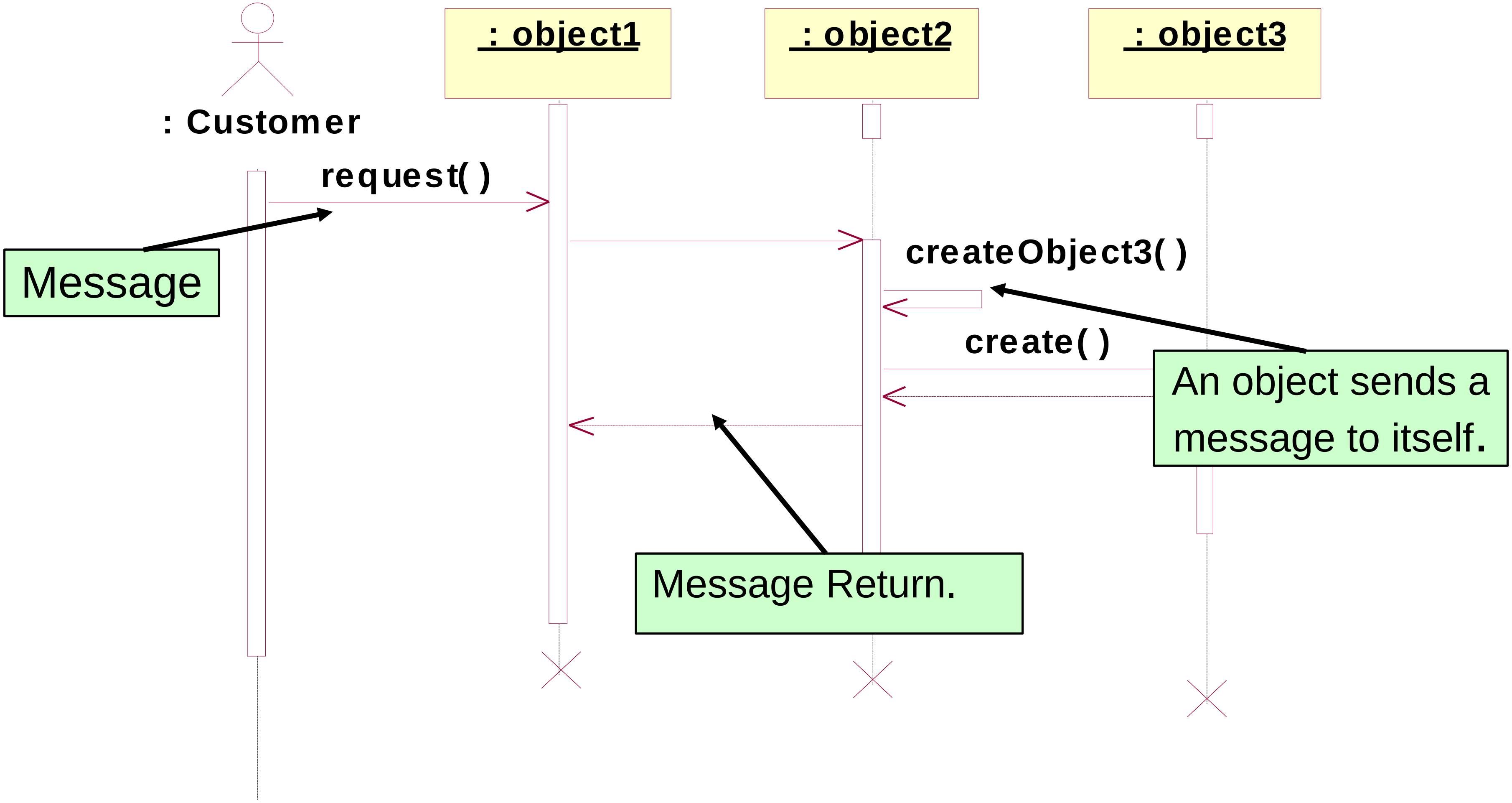
Rectangles refer to events that are related to each other.

Sequence diagrams: messages



Arrows indicate messages that are sent from one object to another.

Sequence diagrams - different types of messages

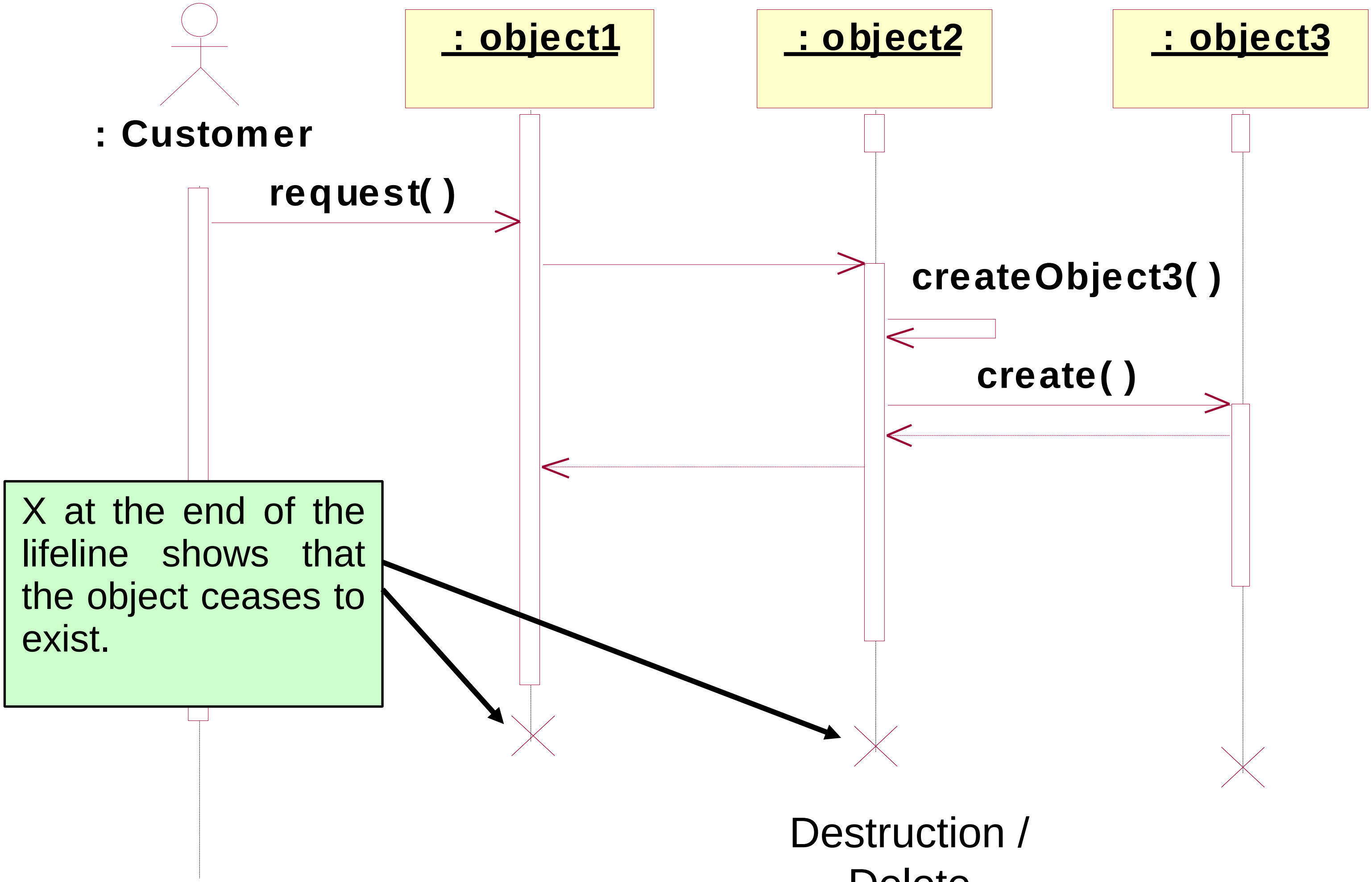


Message

An object sends a message to itself.

Message Return.

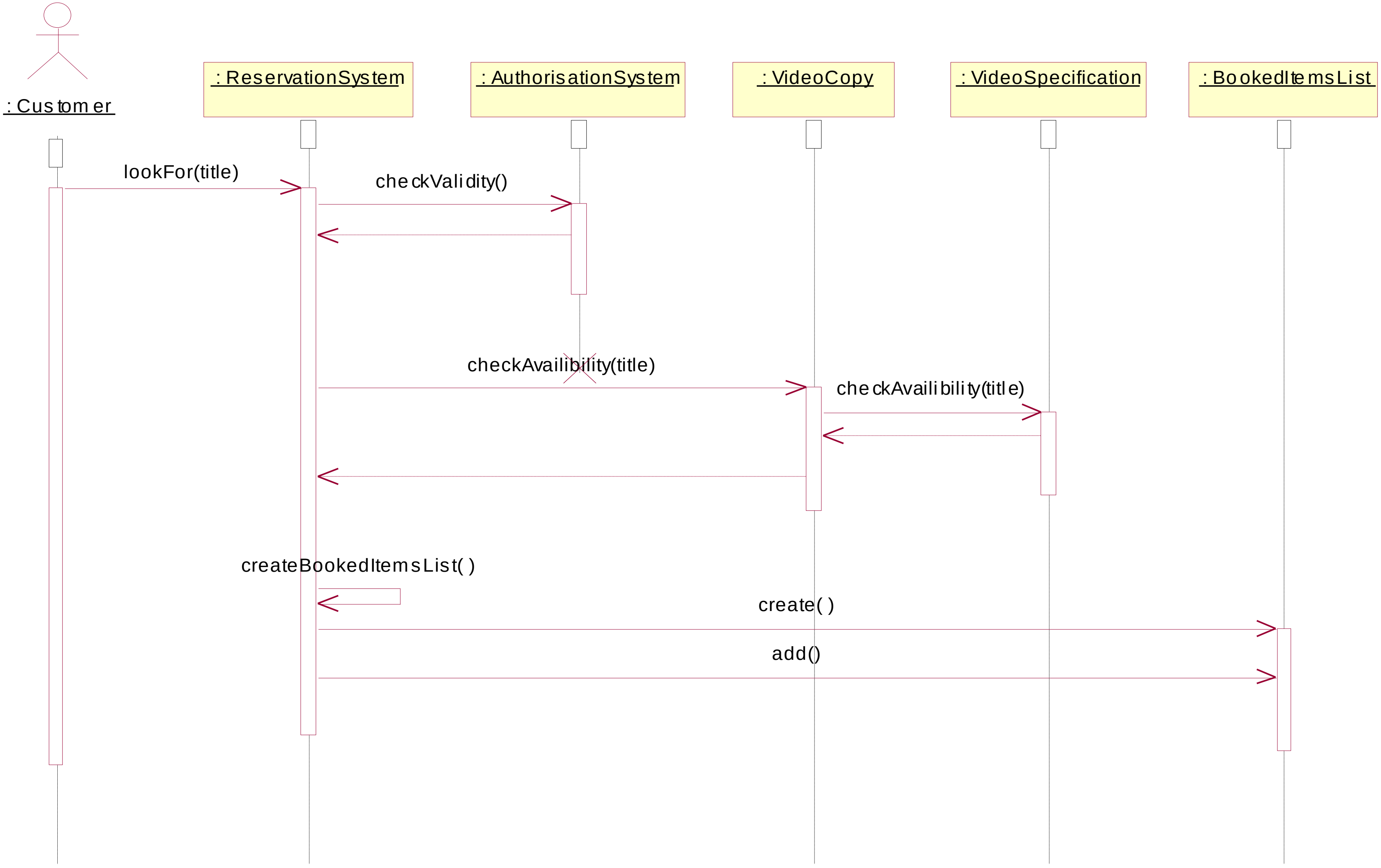
Sequence diagrams: endpoints

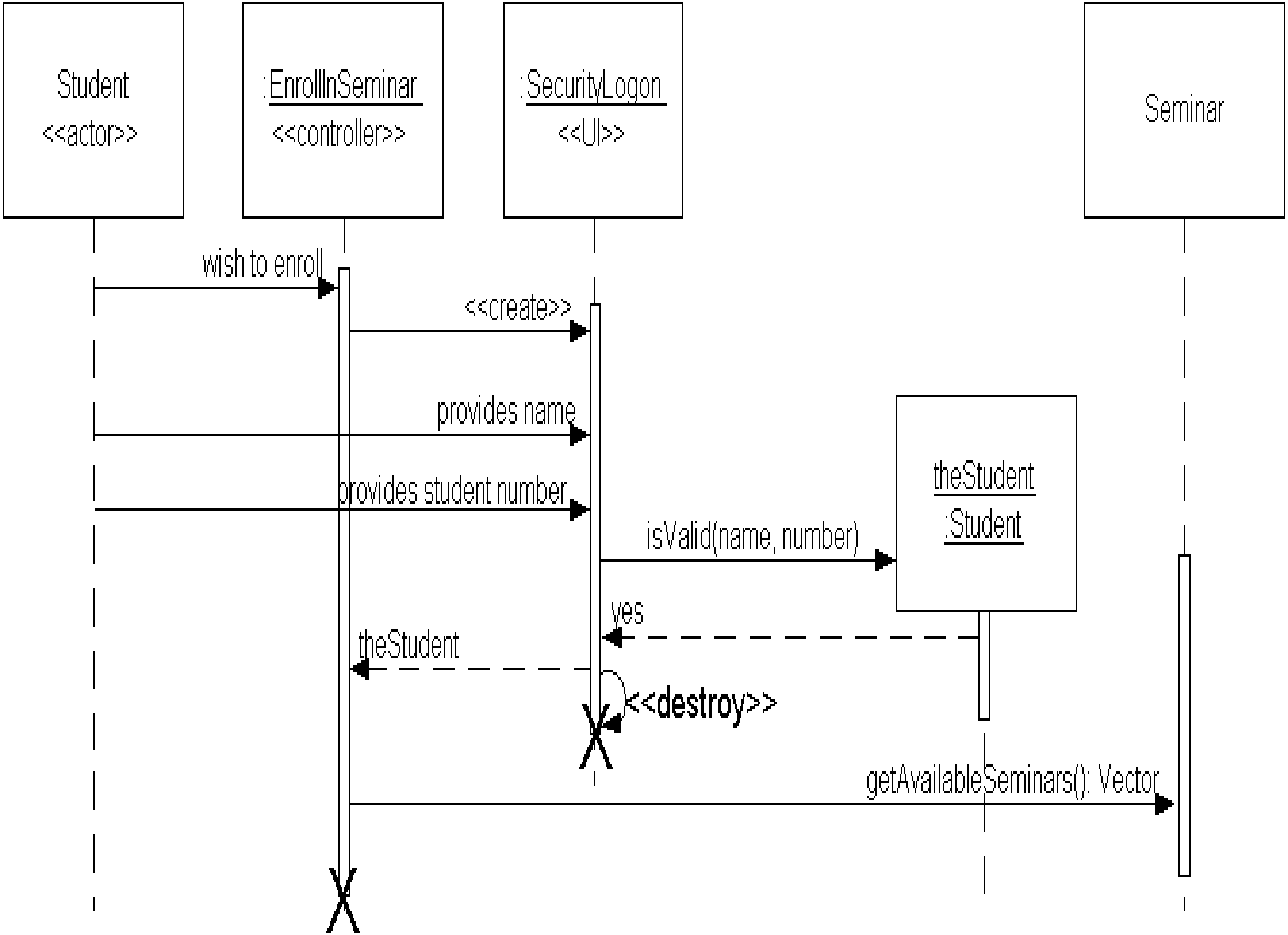


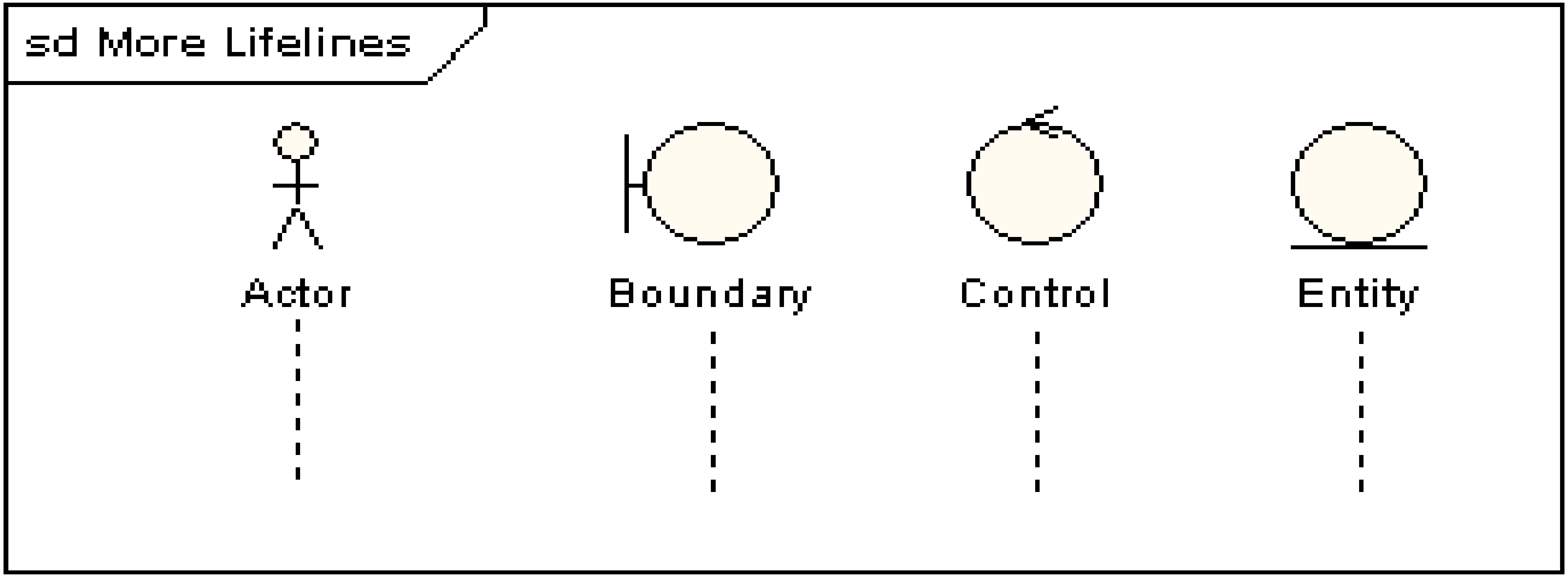
X at the end of the lifeline shows that the object ceases to exist.

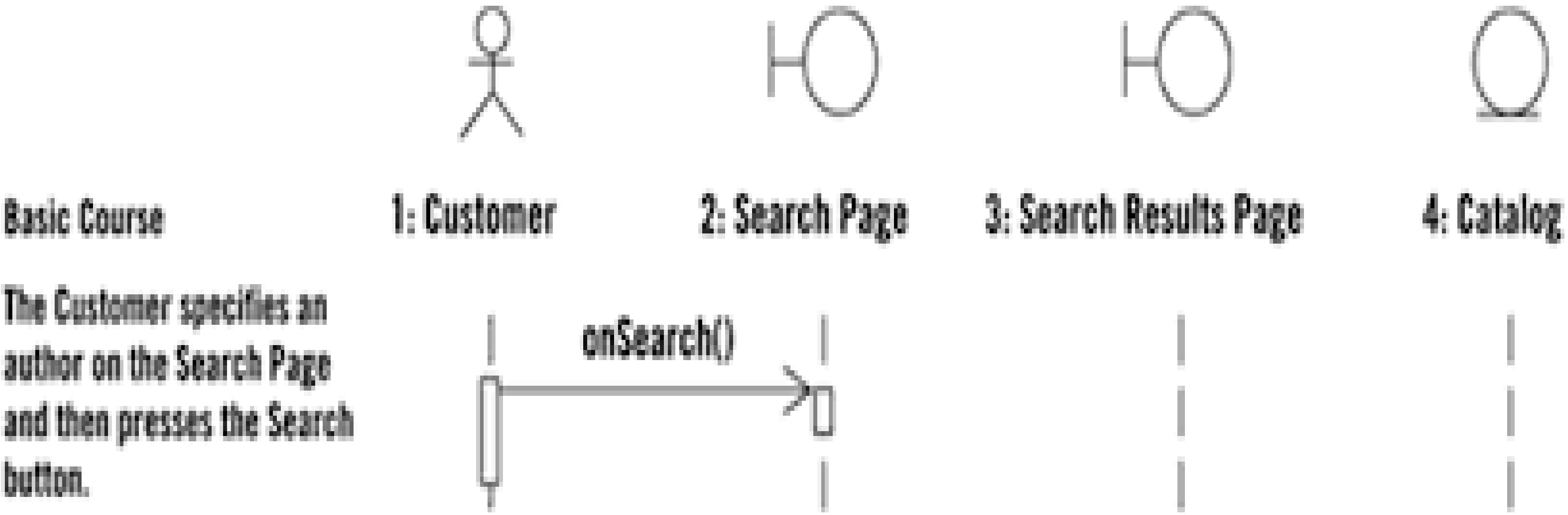
Destruction / Delete

Sequence diagram: a more complex example




Student
<<actor>>
:EnrollInSeminar
<<controller>>
:SecurityLogon
<<UI>>
Seminar
wish to enroll
<<create>>
provides name
provides student number
theStudent
:Student
isValid(name, number)
yes
theStudent
<<destroy>>
getAvailableSeminars(): Vector





Basic Course

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

Checkout an online order.

1. The customer decides to checkout.

...

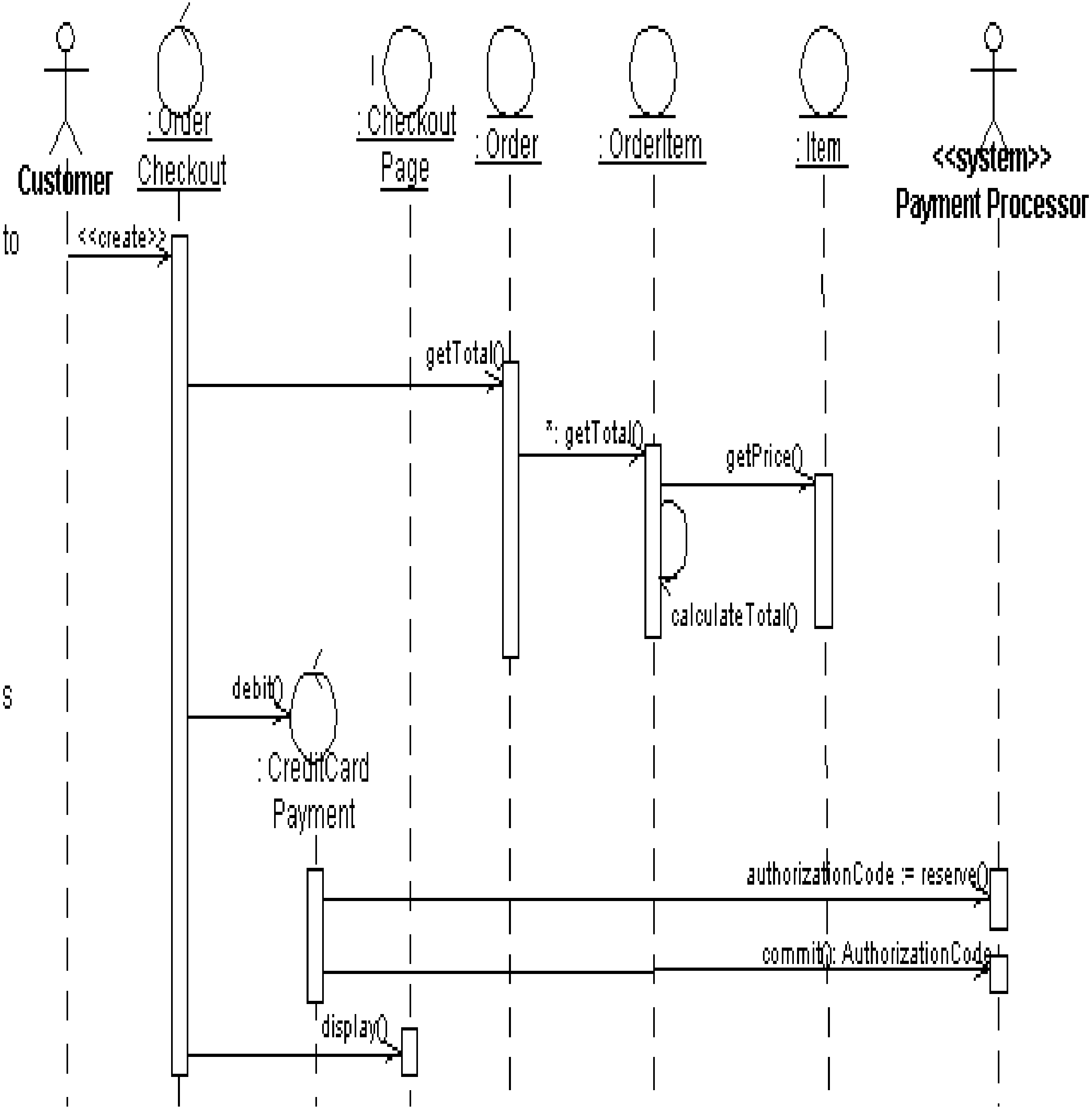
5. The system calculates the order total.

...

12. The system processes the credit card payment.

...

14. The system displays the checkout summary page.



Sequence Diagrams	Collaboration Diagrams
The sequence diagram are used to represent the sequence of messages that are flowing from one object to another.	The collaboration diagram are used to represent the structural organization of the system and the messages that are sent and received.
New object is added to the right. It generally shows the sequence of events that occur.	objects can be placed anywhere on the diagram. It demonstrates how objects are statically connected.
The sequence diagram is used when time sequence is main focus.	The collaboration diagram is used when object organization is main focus.
The sequence diagrams are better suited of analysis activities.	The collaboration diagrams are better suited for depicting simpler interactions of the smaller number of objects.

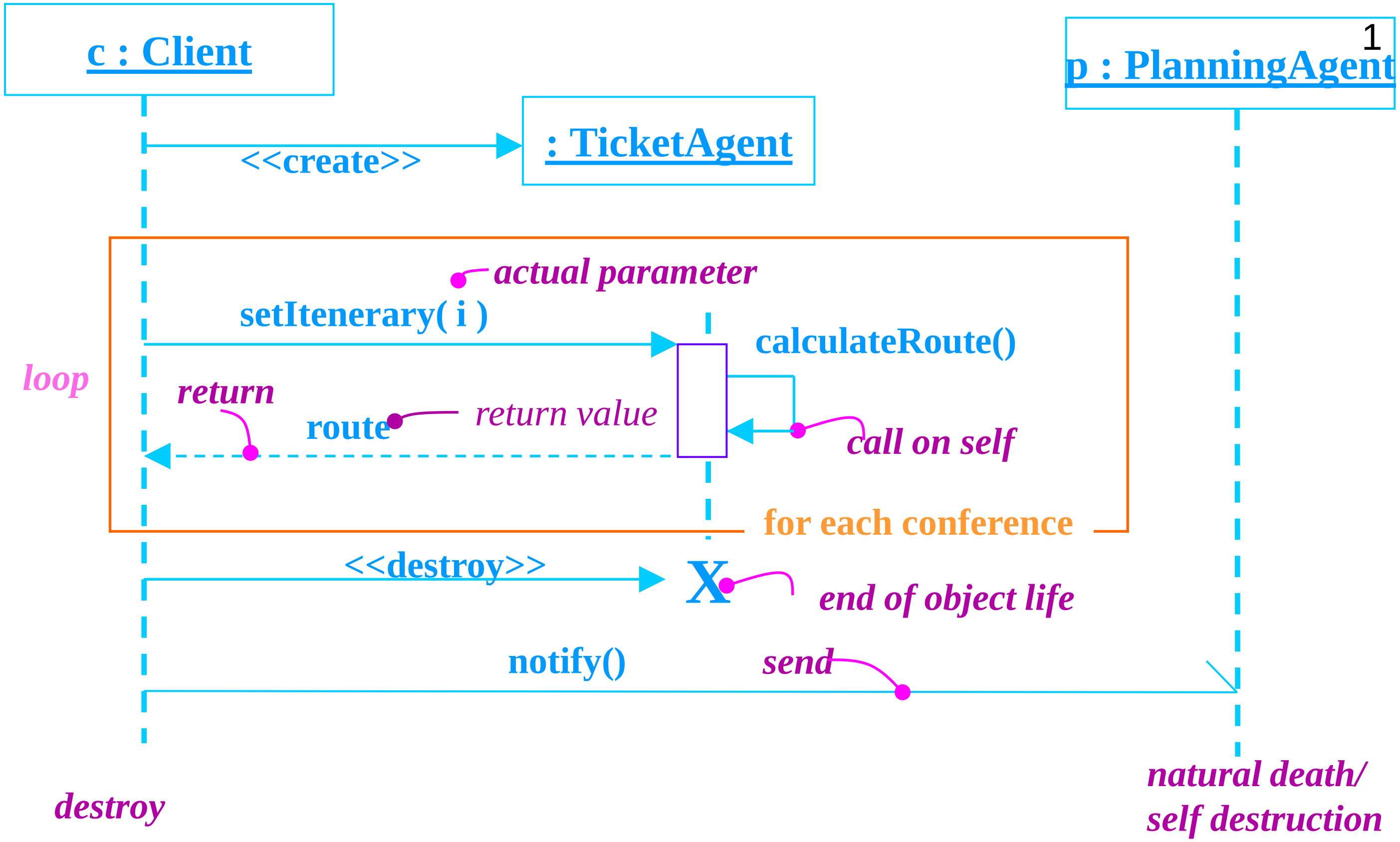
Interactions - Modeling Actions

- Simple →
- Call →
- Return ⇢
- Send ⇀

asynchronous in 2.0 (stick arrowhead) – no return value expected at end of callee activation

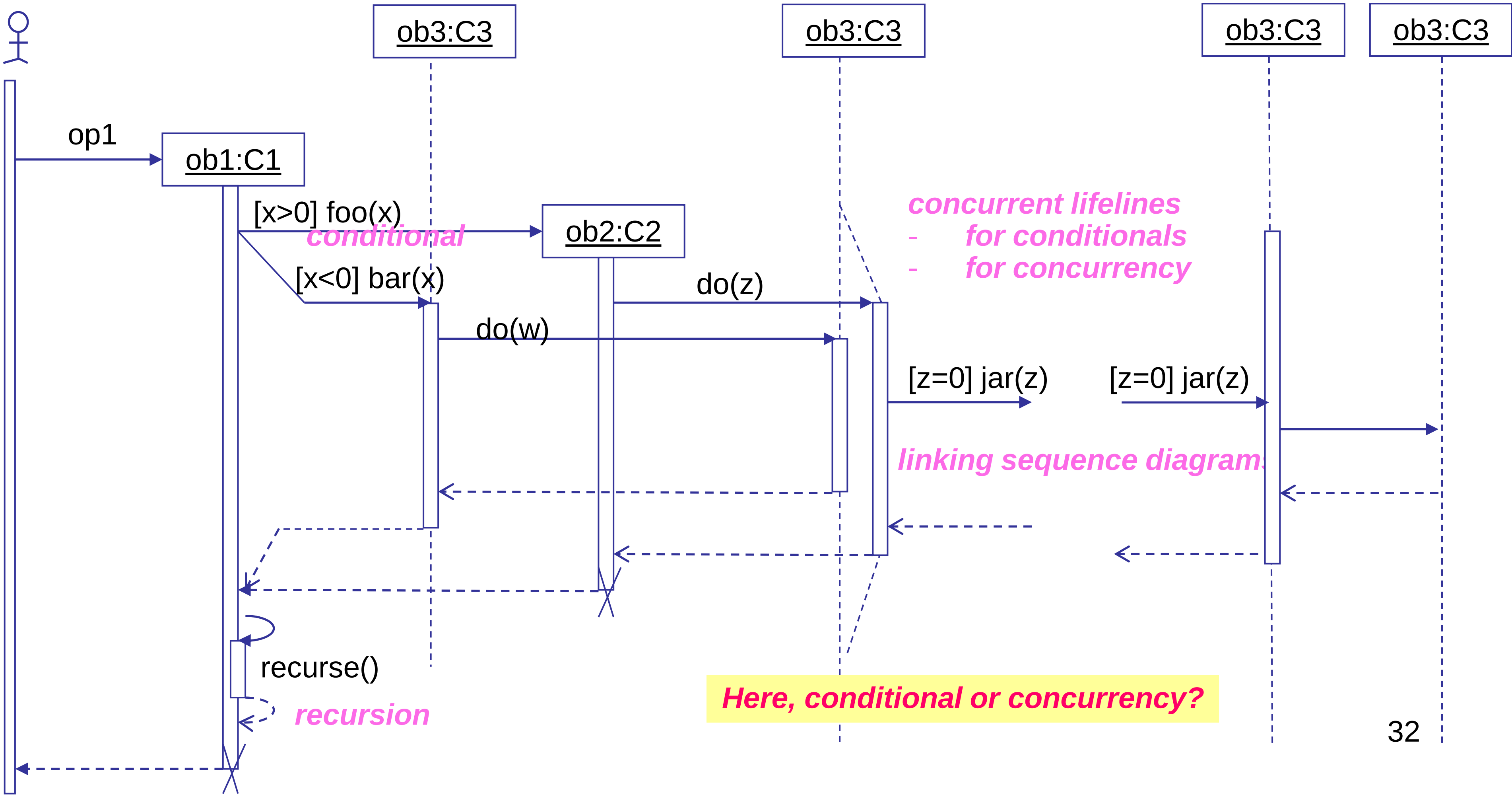
activation of caller may end before callee's
(???)

half arrow in 1.x



Sequence Diagrams – Generic vs. Instance

- 2 forms of sd:
  - **Instance** sd: describes a specific scenario in detail; no conditions, branches or loops.
  - **Generic** sd: a use case description with alternative courses.

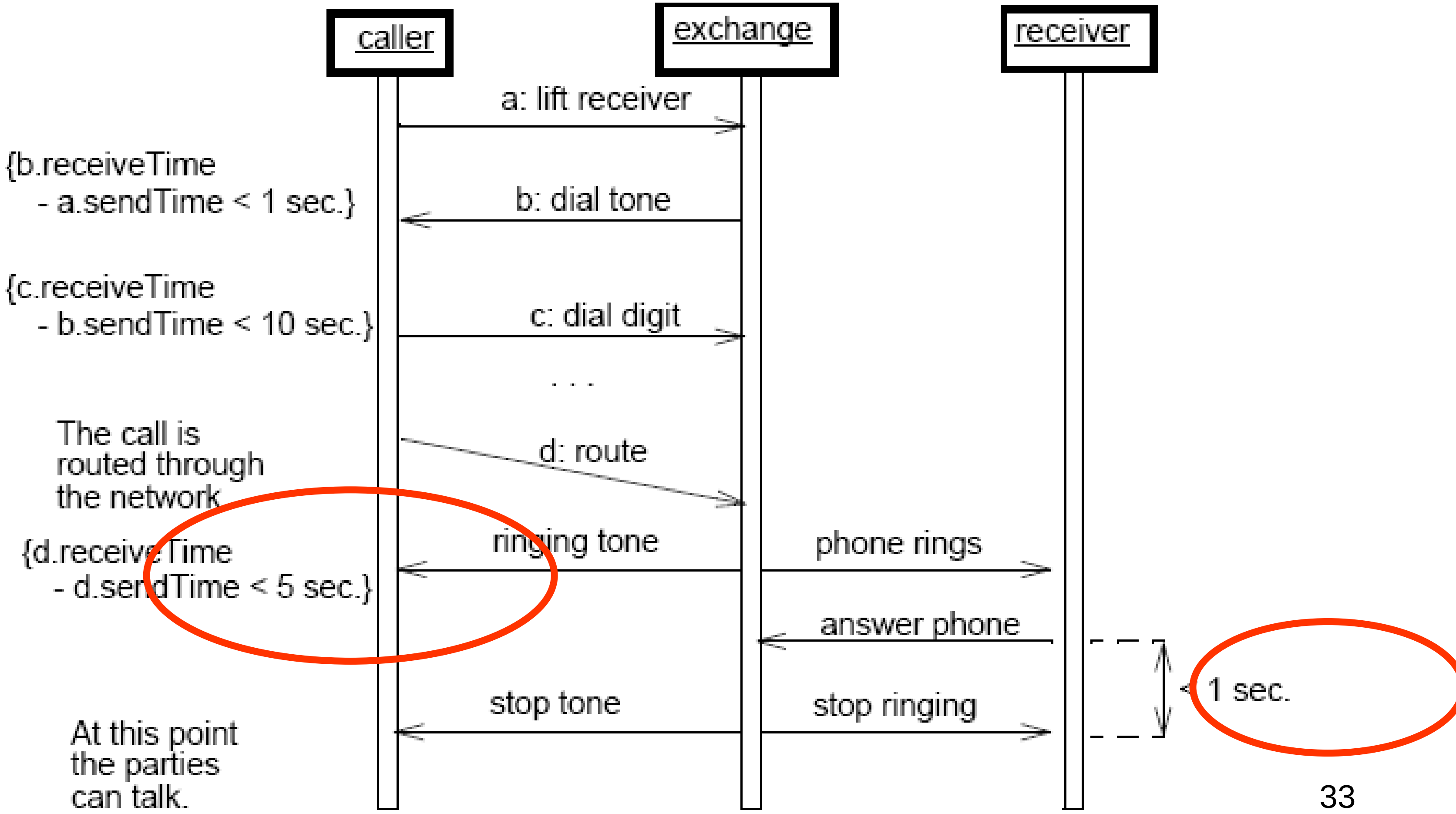


Here, conditional or concurrency?

Timing constraints

- Useful in real-time applications
- useful to specify race condition behaviour
- Two ways to specify (in 1.x)

any example?

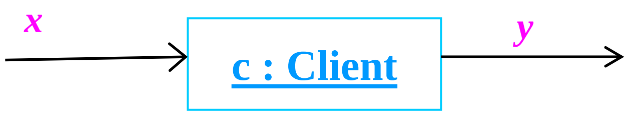


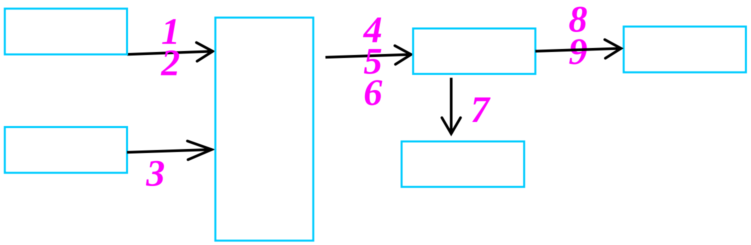
caller
exchange
receiver
a: lift receiver
{b.receiveTime - a.sendTime < 1 sec.}
b: dial tone
{c.receiveTime - b.sendTime < 10 sec.}
c: dial digit
. . .
The call is routed through the network
d: route
{d.receiveTime - d.sendTime < 5 sec.}
ringing tone
phone rings
answer phone
< 1 sec.
stop tone
stop ringing
At this point the parties can talk.

Interactions - Procedural Sequencing vs. Flat Sequencing

Flat Sequencing

- Infrequent: *Not recommended for most situations.*
- Each message is numbered sequentially in order of timing.
- Rendered with stick arrowhead.



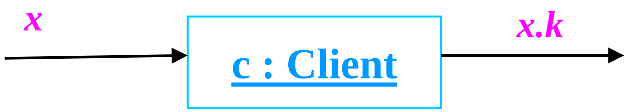


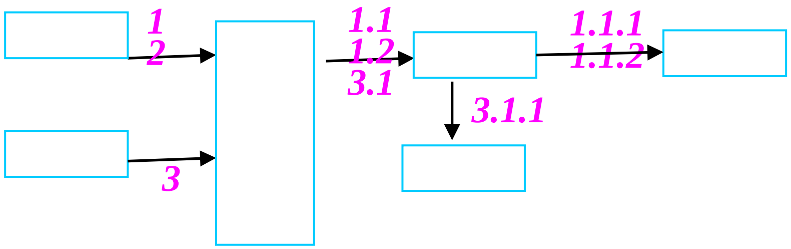
CAN'T TELL RELATIONSHIPS

Procedural Sequencing

(decimal system)

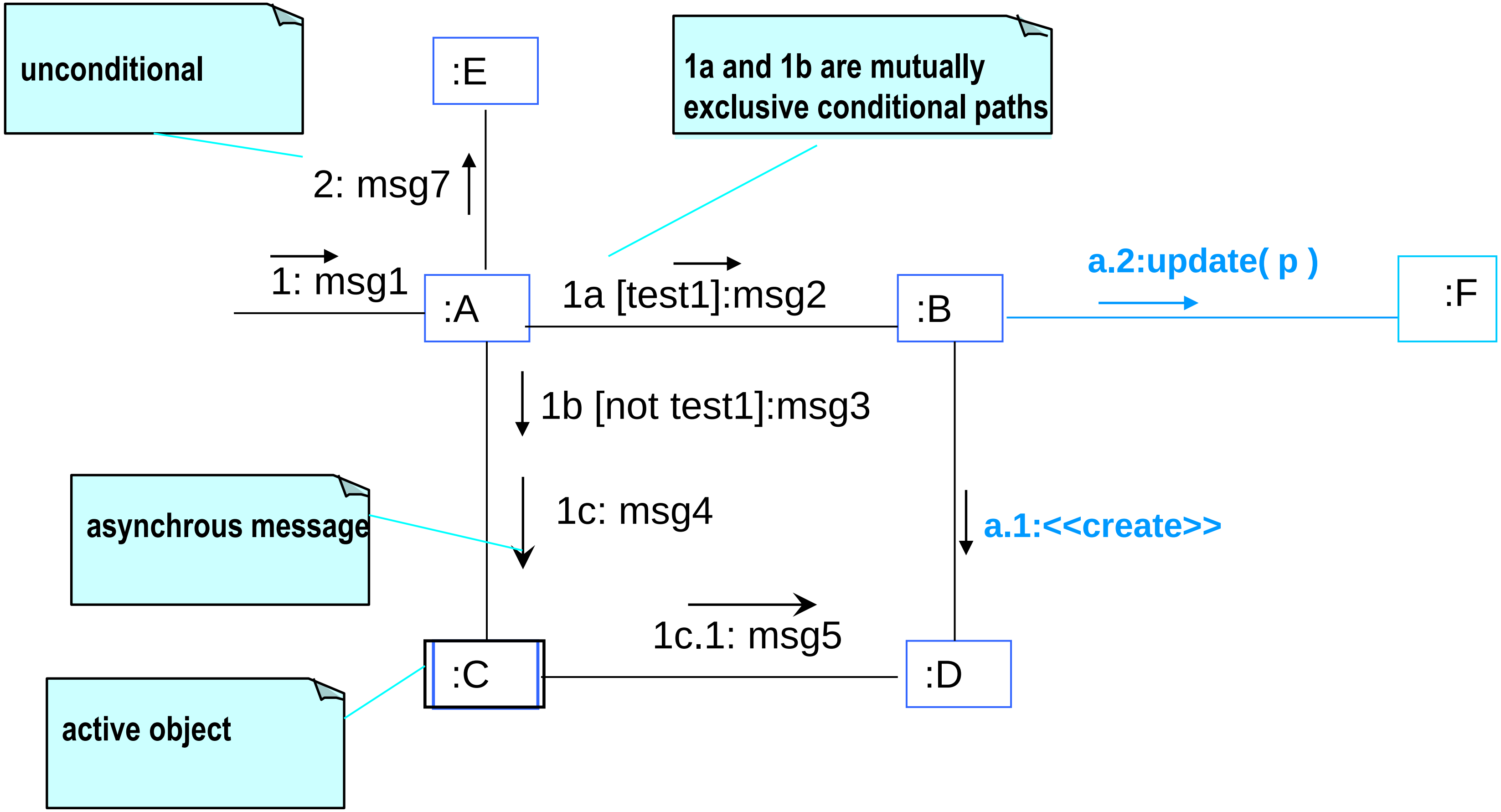
- Most common.
- Each message within the same operation is numbered sequentially.
- Nested messages are prefixed with the sequence number of the invoking operation.
- Rendered with filled solid arrow.





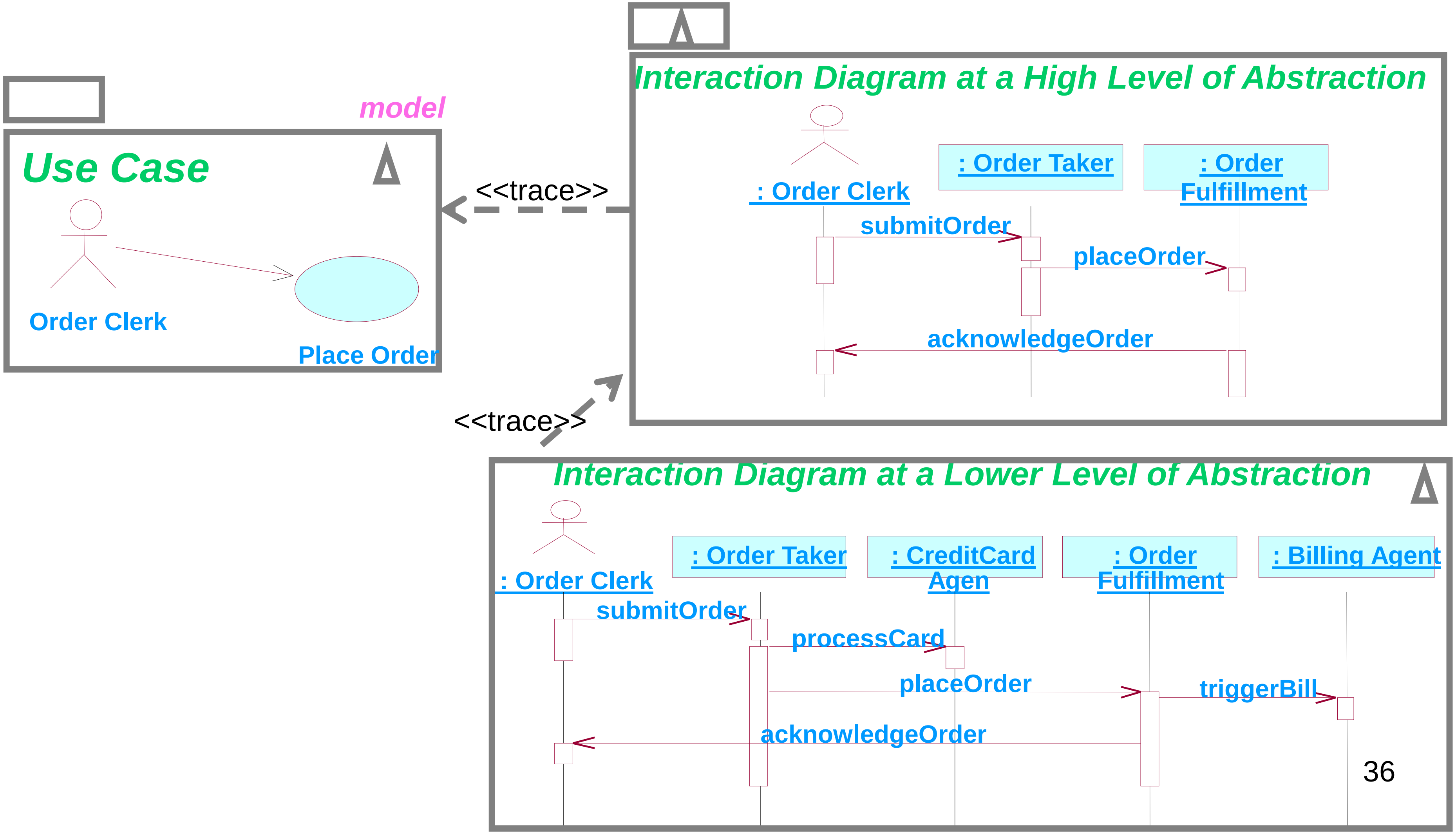
Interactions – conditional paths, asynchronous message

[Craig Larman] [http://www.phptr.com/articles/article.asp?p=360441&seqNum=6&rl=1]

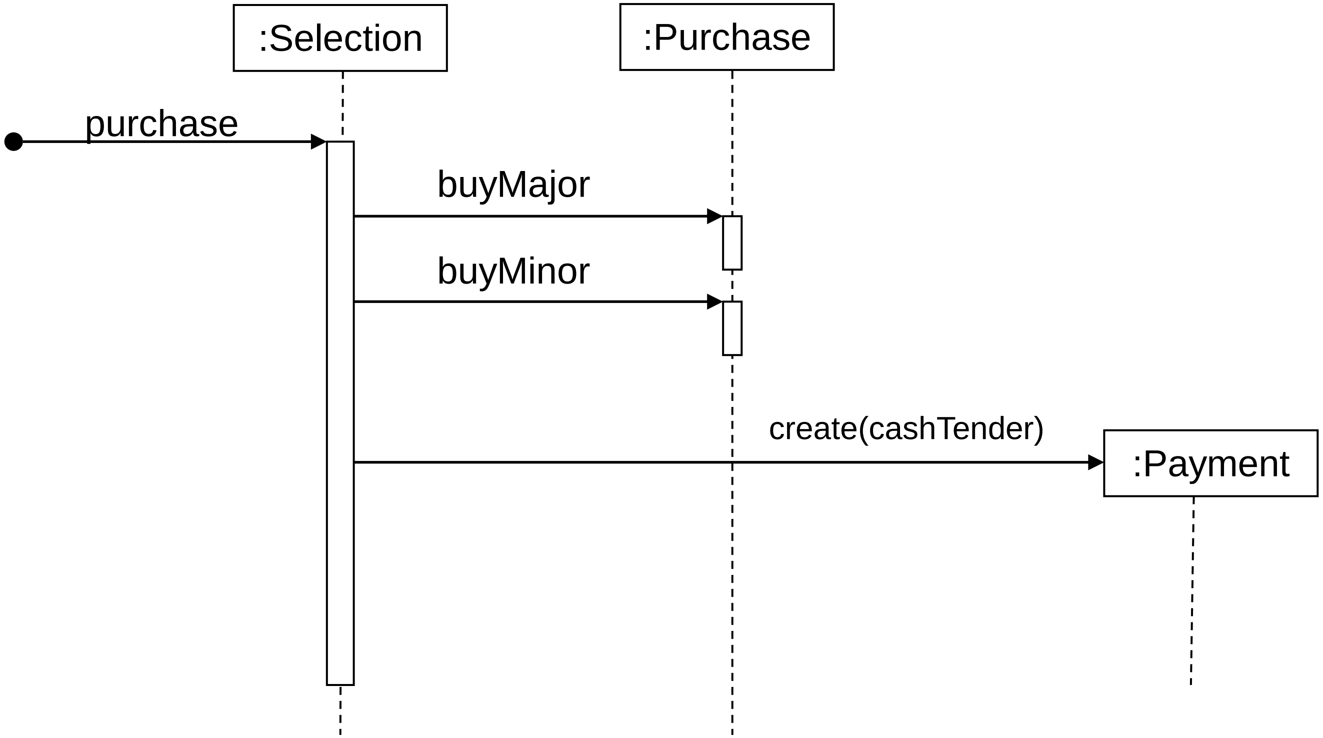


Modeling Different Levels of Abstraction

- Establish trace dependencies between high and low levels of abstraction
- Lo*osely couple different levels of abstraction*
    - ***Use Cases*** trace to ***Collaborations*** in the ***Design*** Model, to a society of ***classes***
    - •***Components*** trace to the elements in the design model, then to ***Nodes***

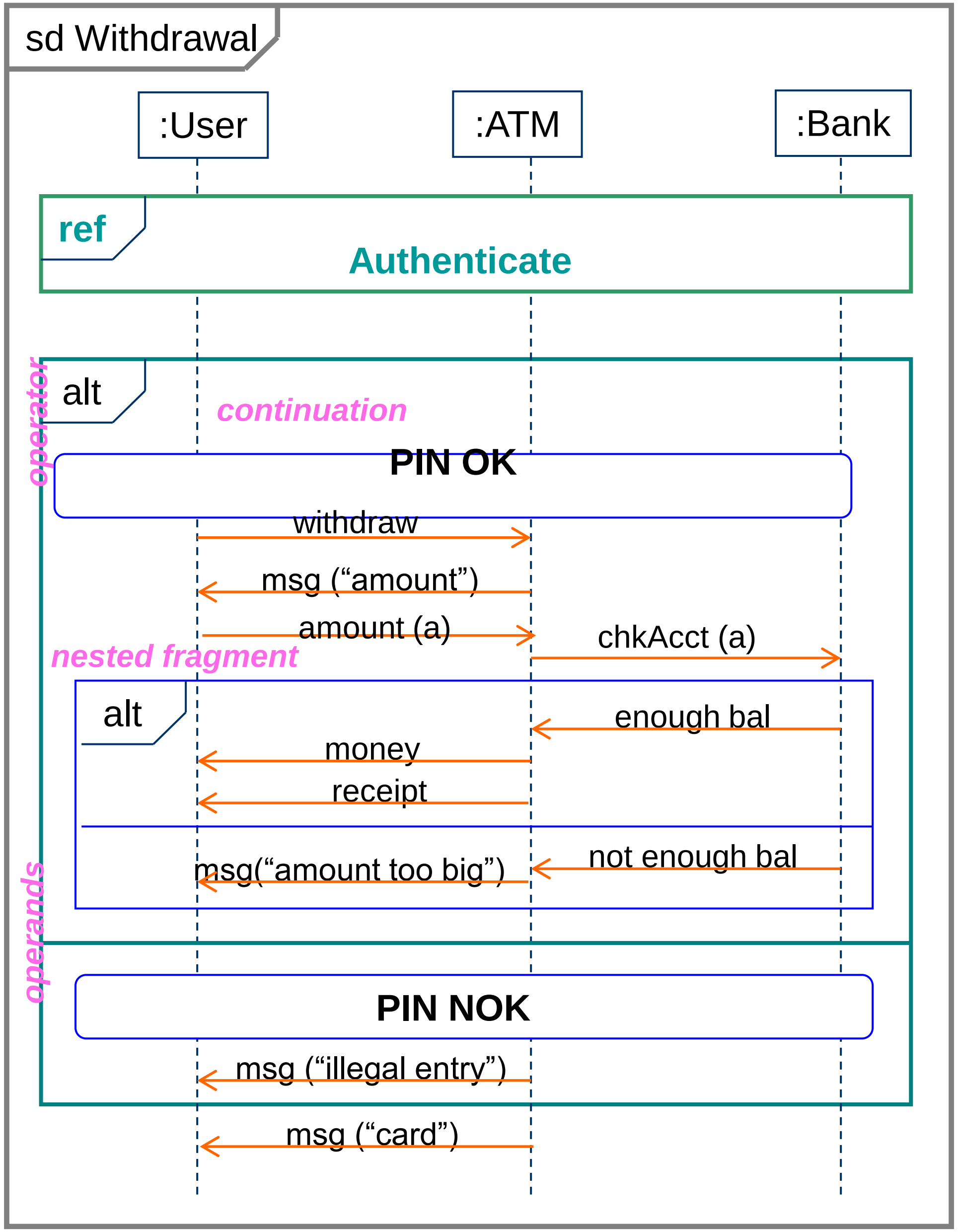


Sequence Diagrams & Some Programming

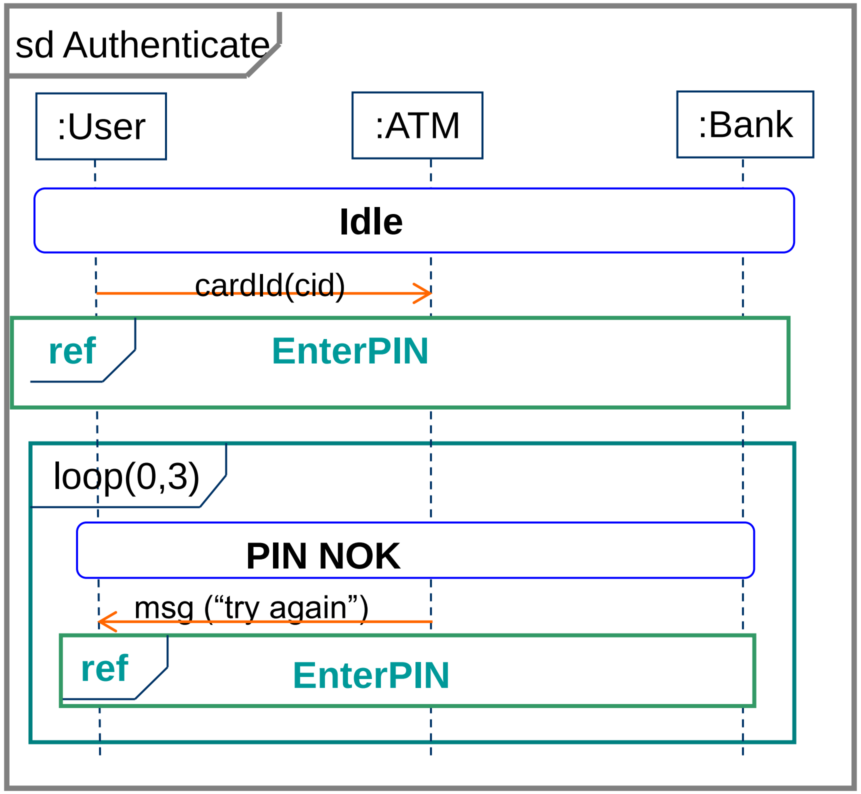


```
public Class Selection
            { private Purchase myPurchase = new Purchase();
              private Payment myPayment;
               public void purchase()
                        { myPurchase.buyMajor();
                          myPurchase.buyMinor():
                          myPayment = new Payment( cashTender );
                          //. .
                        }
              // . .
            }
```

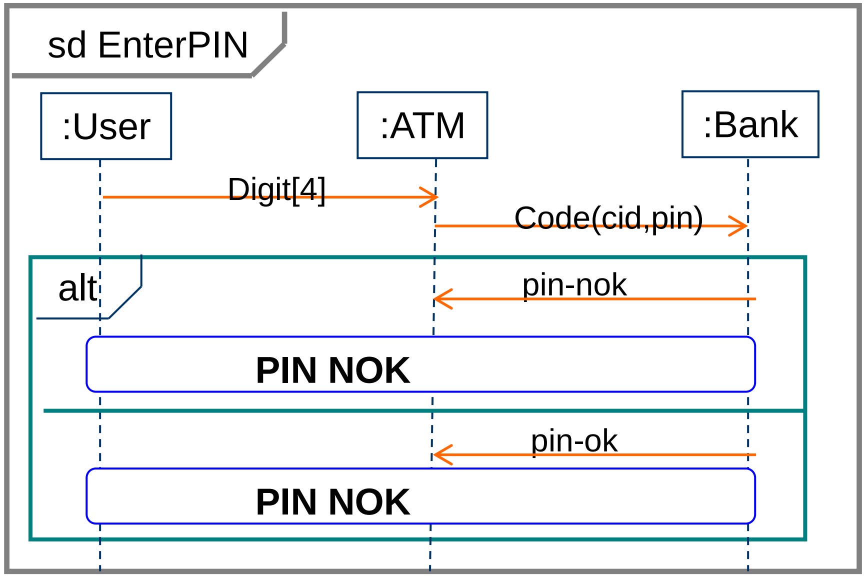
Frames: References



sd Withdrawal
:User
:ATM
:Bank
ref
Authenticate
operator
alt
continuation
PIN OK
withdraw
msg (“amount”)
amount (a)
chkAcct (a)
nested fragment
alt
enough bal
money
receipt
msg(“amount too big”)
not enough bal
operands
PIN NOK
msg (“illegal entry”)
msg (“card”)

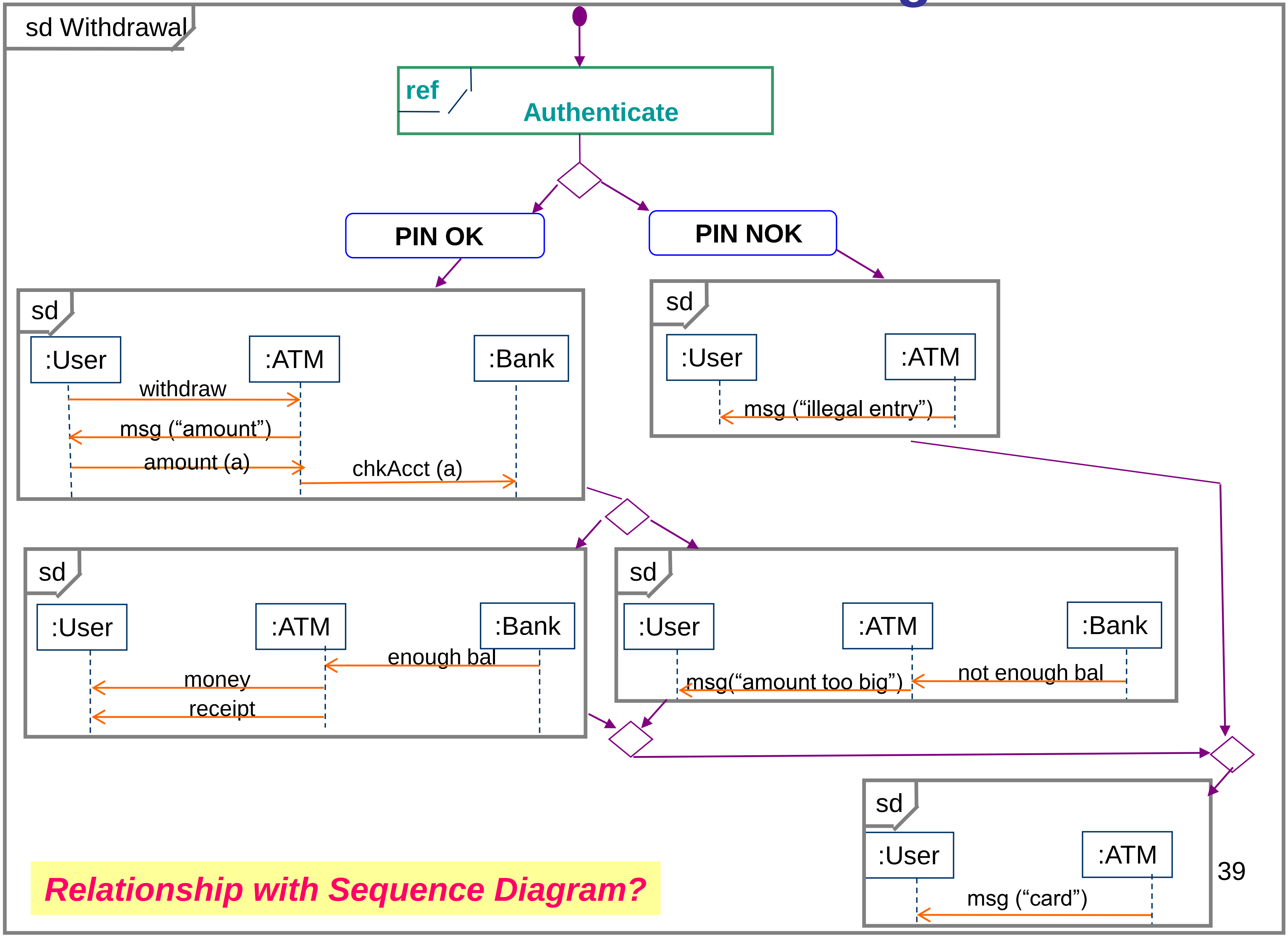


sd Authenticate
:User
:ATM
:Bank
Idle
cardId(cid)
ref
EnterPIN
loop(0,3)
PIN NOK
msg (“try again”)
ref
EnterPIN



sd EnterPIN
:User
:ATM
:Bank
Digit[4]
Code(cid,pin)
alt
pin-nok
PIN NOK
pin-ok
PIN NOK

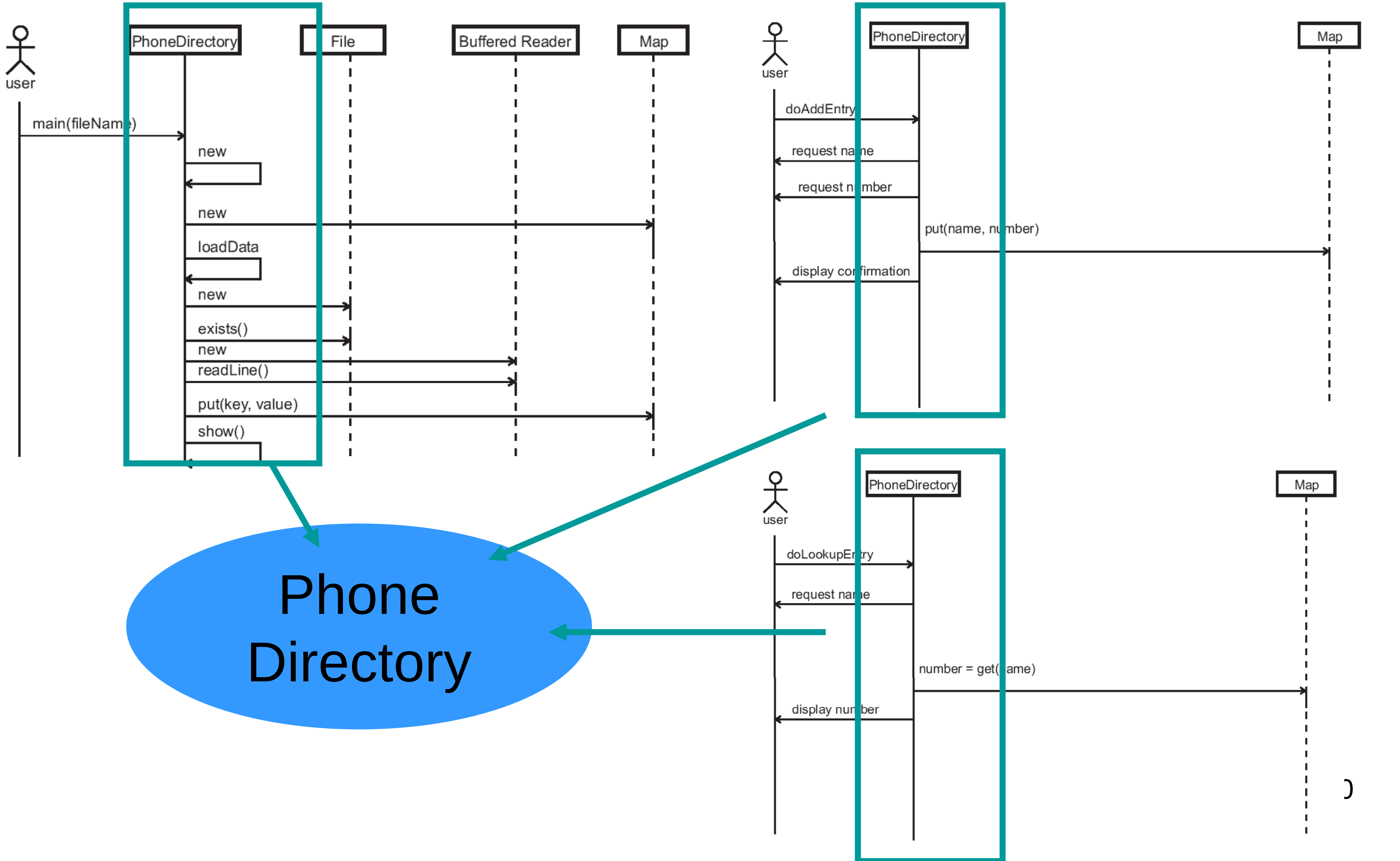
Interaction Overview Diagram



Relationship with Sequence Diagram?

From Diagrams to Objects

Collect all messages to define object’s methods and state transitions !



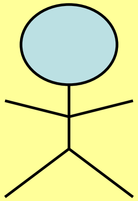
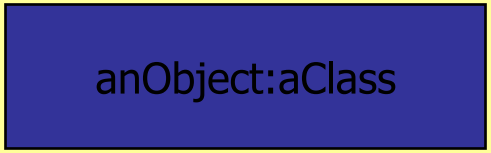
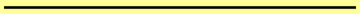
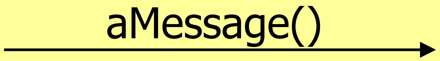
Collaboration Diagram

- Represents a Collaboration of Objects and Interaction
- Collaboration diagrams, also known as **communication diagrams**, are a type of Unified Modeling Language (UML) diagram used in software engineering **to illustrate and visualize the dynamic interactions and collaborations between objects within a system**
- Collaboration diagrams focus on representing the **flow of messages exchanged between objects during specific scenarios or interactions**.
- Provide a dynamic view of a system, emphasizing the communication patterns and relationships among various objects. Collaboration diagrams are particularly valuable for capturing real-time scenarios and highlighting the temporal aspects of object interactions.
- Play a crucial role in communication among development team members and aid in the **analysis and design phases** of software development**.**

Steps to Create a UML Collaboration Diagram

- Identify Objects
- Determine Relationships
- Define Messages
- Draw the Diagram
    - Objects
    - Messages
    - Relationships **(Links)**
- Arrange Objects
- Label Messages
- Add Notations
- Review and Refine
- Document
- Validation
- Feedback and Iteration
- Finalize

Collaboration Diagram Syntax

AN ACTOR	
AN OBJECT	 anObject:aClass
AN ASSOCIATION	
A MESSAGE	 aMessage()

Objects

- **Objects** rectangles containing the object
- **Signatureobject :**
  - **object name : object Class**
  - object name (optional) - starts with lowercase letter
  - class name (mandatory) - starts with uppercase letter
- Objects connected by lines **actor** can appear
- Objects participating in a collaboration come in two flavors—**supplier and client**

- **Supplier objects** are the objects that supply the method that is being called, and therefore **receive** the message
- **Client objects call methods on supplier objects,** and therefore **send** messages

Links

- The connecting lines drawn between objects are links
- They enable you to see the relationships between objects
- This symbolizes the ability of objects to send messages to each other
- A single link can support one or more messages sent between objects

Messages

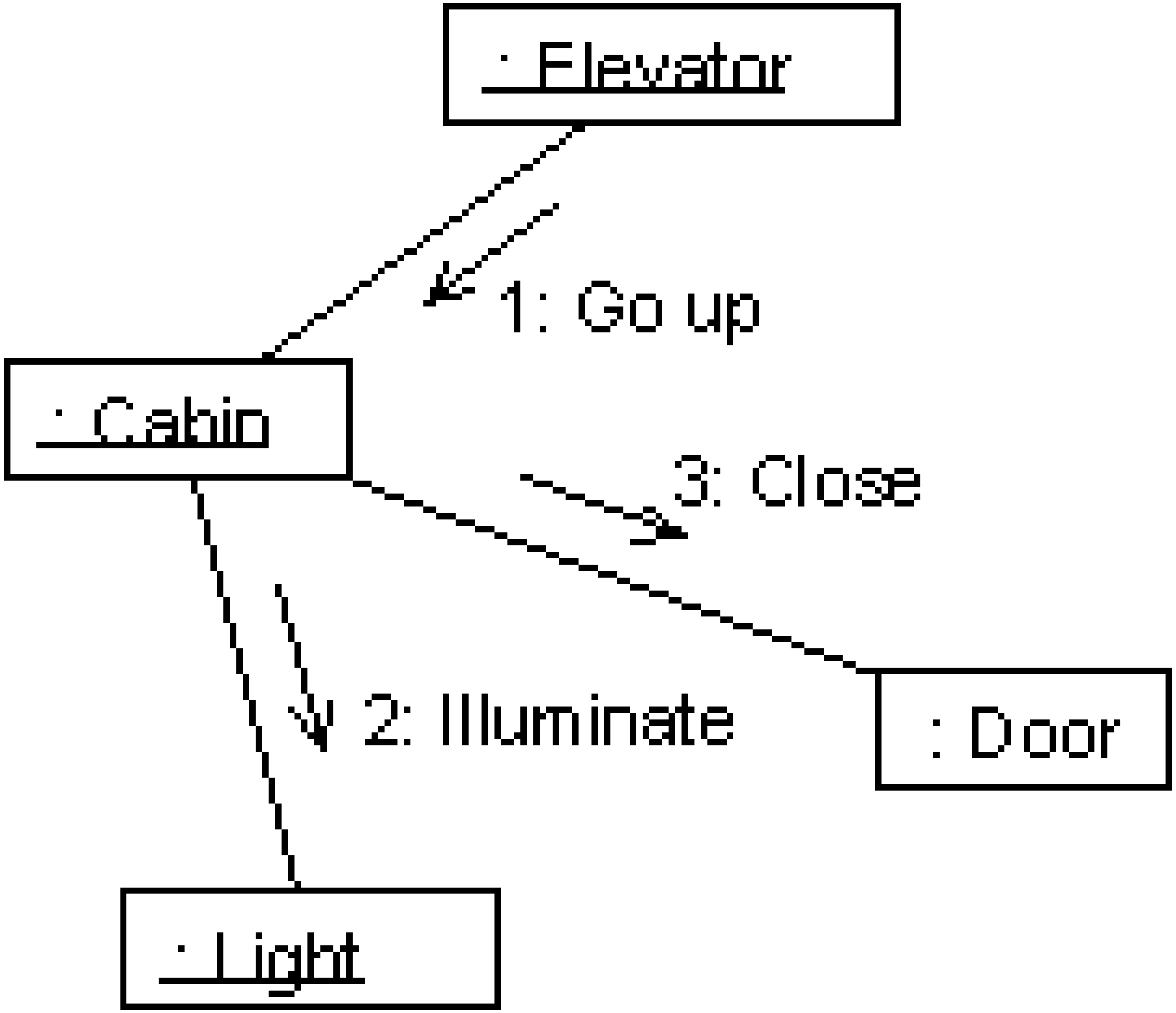
- An interaction is implemented by a group of objects that collaborate by exchanging messages
- Messages in collaboration diagrams are shown as arrows pointing from the Client object to the Supplier object.
- Typically, messages represent a client invoking an operation on a supplier object
- Message icons have one or more messages associated with them
- Messages are composed of message text prefixed by a sequence number
- Time is not represented explicitly in a collaboration diagram, and as a result the various messages are numbered to indicate the sending order

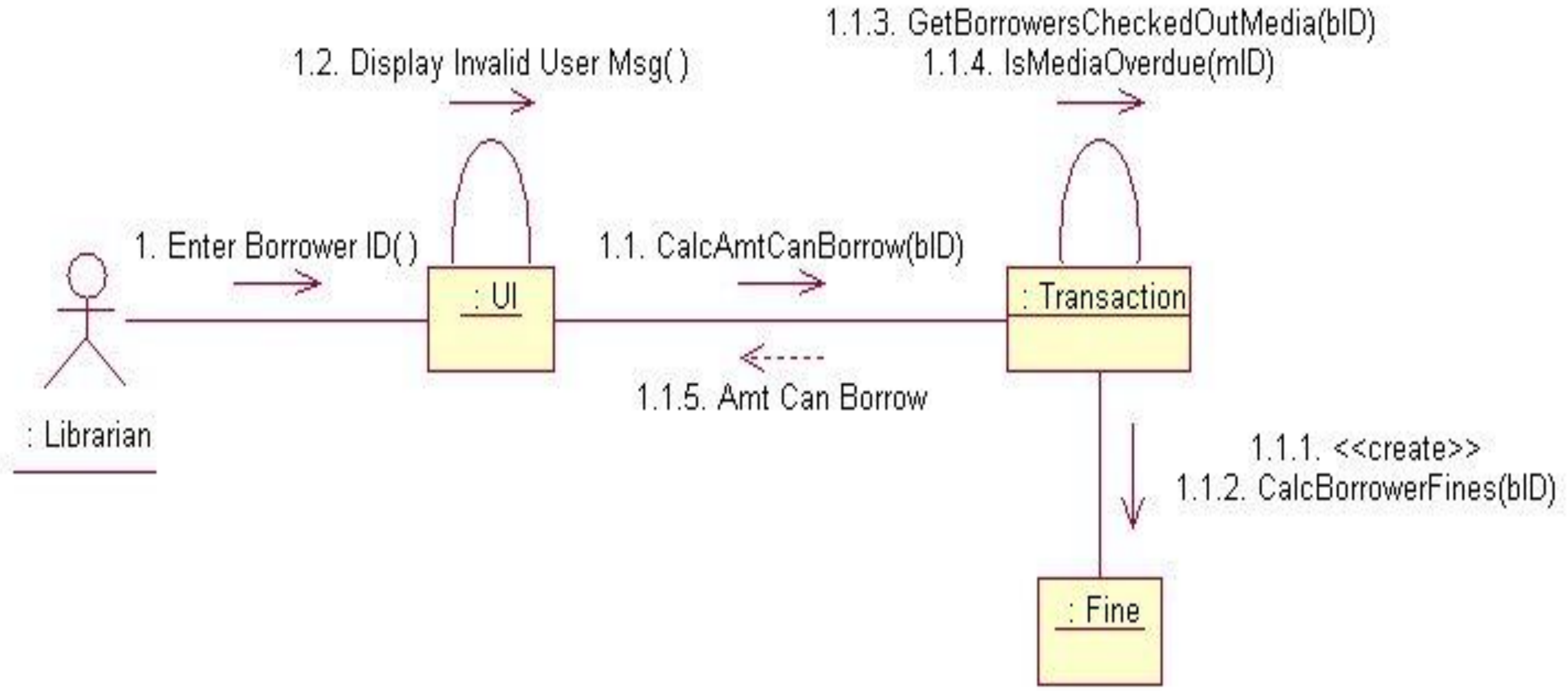
Steps to Creating a Collaboration Diagram

Determine the scope of the diagram- the use case it relates to

1. Place the objects that participate in the collaboration on the diagram
   - Remember to place the most important objects towards the center of the diagram.
2. If a particular object has a property or maintains a state that is important to the collaboration, **set the initial value of the property or state**
4. Create links between the objects
5. Create messages associated with each link
6. Add sequence numbers to each message corresponding to the time-ordering of messages in the collaboration

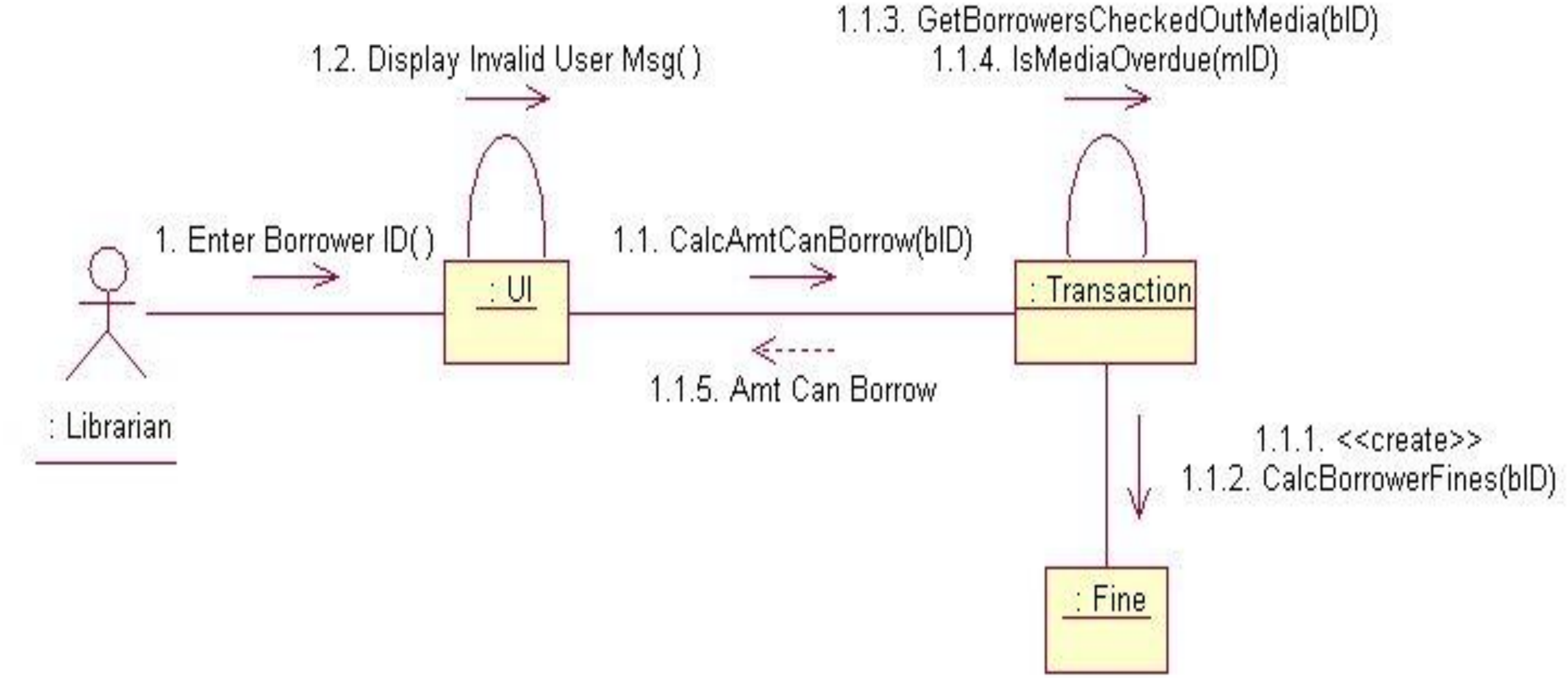
Ex. Collaboration Diagram




1.2. Display Invalid User Msg( )
1.1.3. GetBorrowersCheckedOutMedia(bID)
1.1.4. IsMediaOverdue(mID)
1. Enter Borrower ID( )
1.1. CalcAmtCanBorrow(bID)
: UI
: Transaction
1.1.5. Amt Can Borrow
: Librarian
1.1.1. <<create>>
1.1.2. CalcBorrowerFines(bID)
: Fine

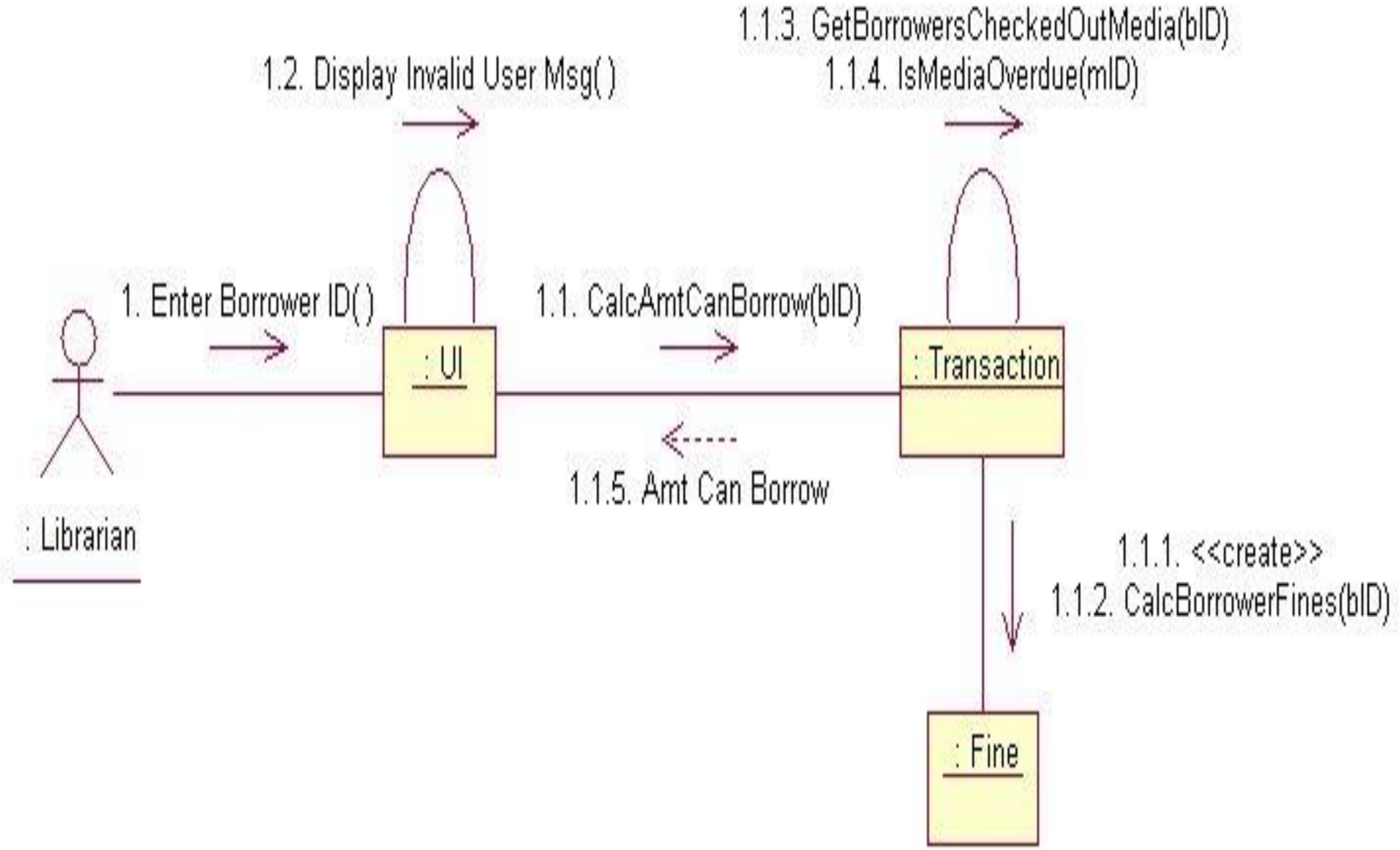
Transaction object acts as a Supplier to the UI (User Interface) Client object.

In turn, the Fine object is a Supplier to the Transaction Client object.


1.2. Display Invalid User Msg( )
1.1.3. GetBorrowersCheckedOutMedia(bID)
1.1.4. IsMediaOverdue(mID)
1. Enter Borrower ID( )
: UI
1.1. CalcAmtCanBorrow(bID)
: Transaction
1.1.5. Amt Can Borrow
: Librarian
1.1.1. <<create>>
1.1.2. CalcBorrowerFines(bID)
: Fine

The visual representation of a link is a straight line between two objects. If an object sends messages to itself, the link carrying these messages is represented as a loop icon. This loop can be seen on both the UI object and the Transaction object.

Flow by Number


1.1.3. GetBorrowersCheckedOutMedia(bID)
1.1.4. IsMediaOverdue(mID)
1.2. Display Invalid User Msg( )
1. Enter Borrower ID( )
1.1. CalcAmtCanBorrow(bID)
: UI
: Transaction
1.1.5. Amt Can Borrow
: Librarian
1.1.1. <<create>>
1.1.2. CalcBorrowerFines(bID)
: Fine

Flow by Numbers

1. Enter Borrower ID

1.1 CalcAmtCanBorrow

1.1.1 <<create>>

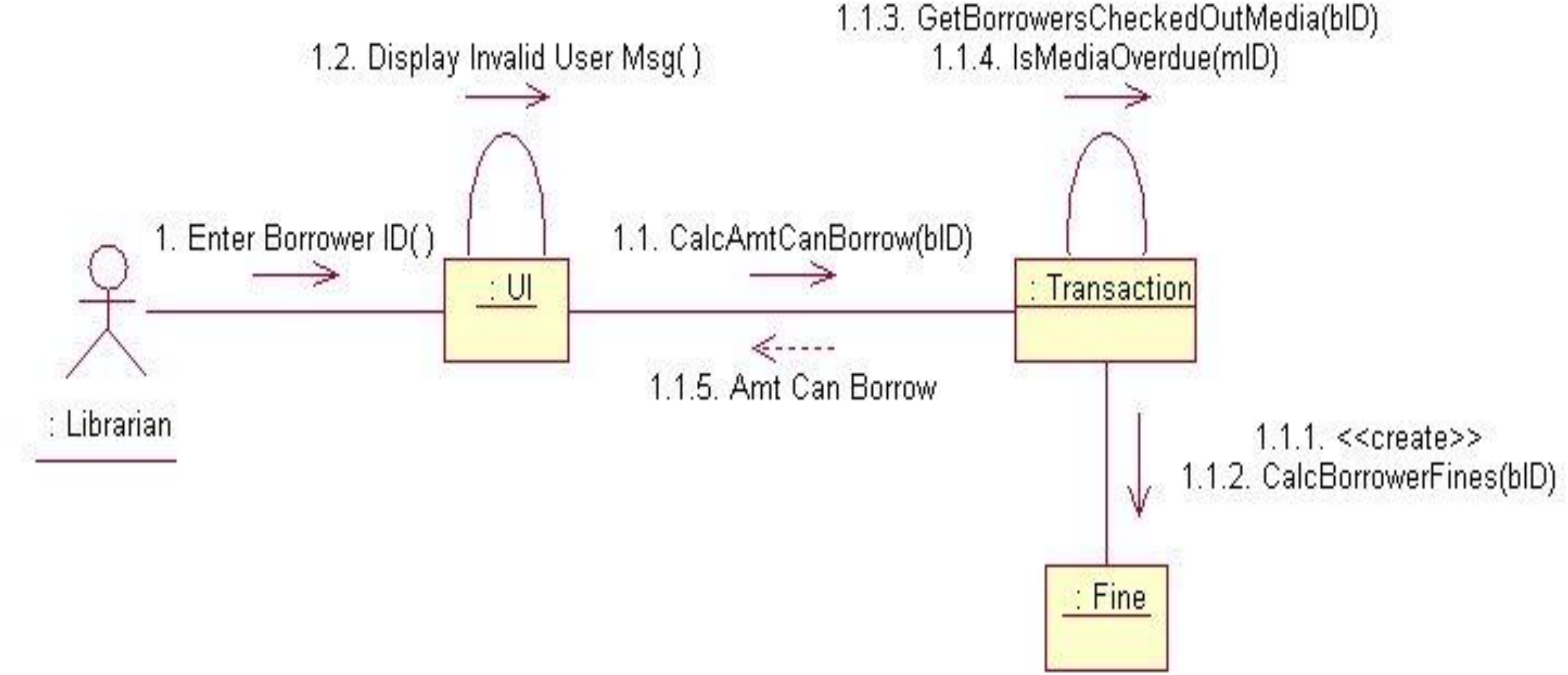
1.1.2 CalcBorrowerFines

1.1.3 GetBorrowersCheckedOutMedia

1.1.4 IsMediaOverdue

1.1.5 Amt Can Borrow

1.2 Display Invalid User Msg

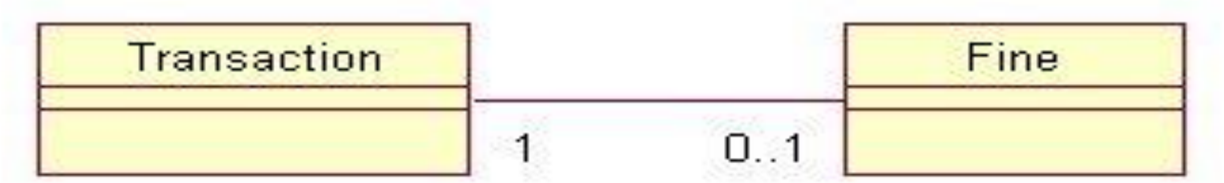

1.1.3. GetBorrowersCheckedOutMedia(bID)
1.1.4. IsMediaOverdue(mID)
1.2. Display Invalid User Msg( )
1. Enter Borrower ID( )
: UI
1.1. CalcAmtCanBorrow(bID)
: Transaction
1.1.5. Amt Can Borrow
: Librarian
1.1.1. <<create>>
1.1.2. CalcBorrowerFines(bID)
: Fine

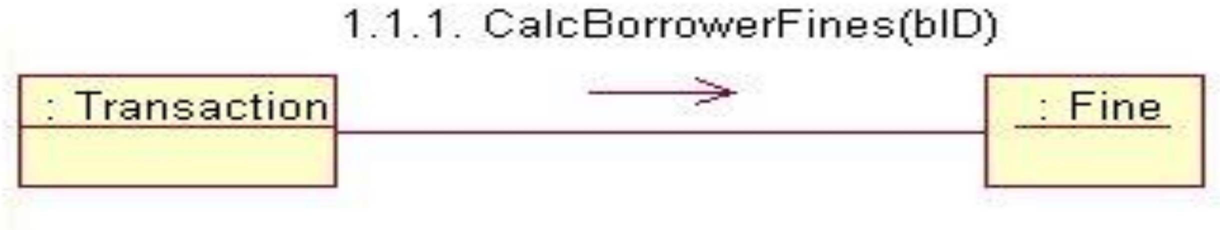
Collaboration diagram is better at showing the relationship between objects

Number of Messages

- In sequence diagrams, each message icon represents a single message.
- In collaboration diagrams, a message icon can represent one or more messages.
- Notice between the Transaction and Fine objects - there is a single message icon, but there are two messages (1.1.1 and 1.1.2) associated with the icon.

Collaboration and Class Diagrams





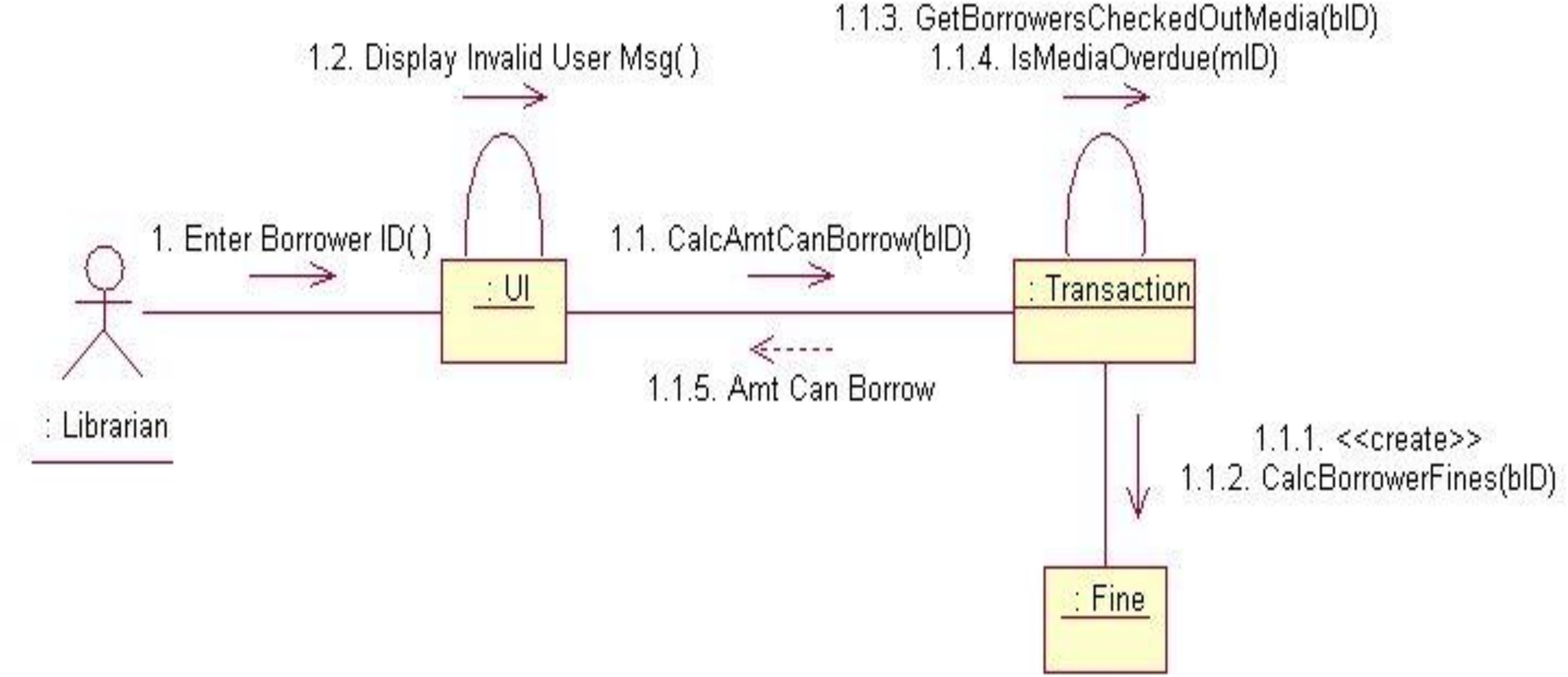
- Links in a collaboration diagram directly correlate to associations between classes in a class diagram

Creation and Deletion

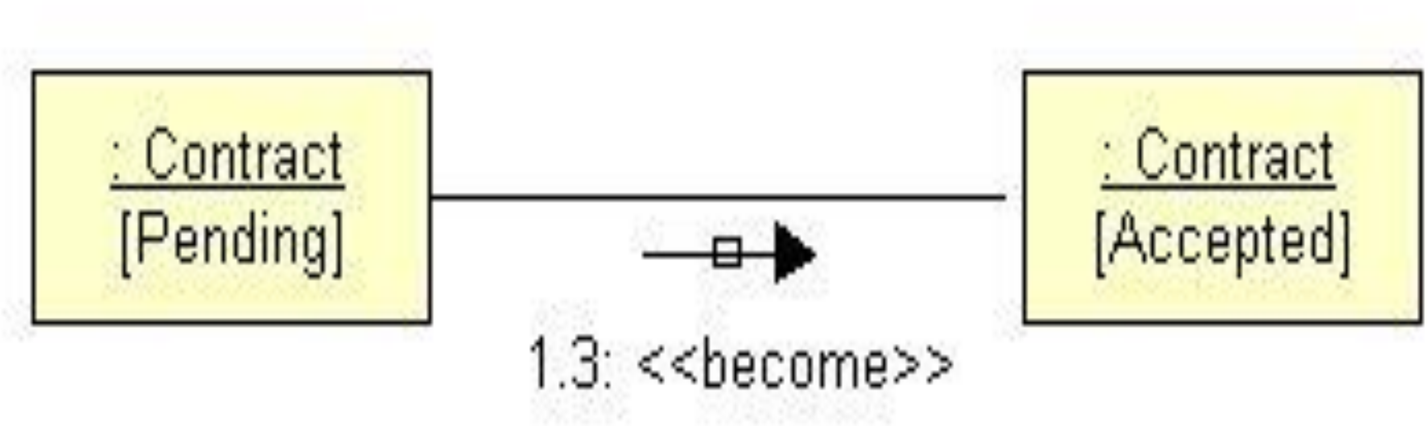
- Unlike sequence diagrams, you don't show an object's lifeline in a collaboration diagram
- If you want to indicate the lifespan of an object in a collaboration diagram, you can use **create and destroy messages** to show when an object is instantiated and destroyed.

Objects Changing State

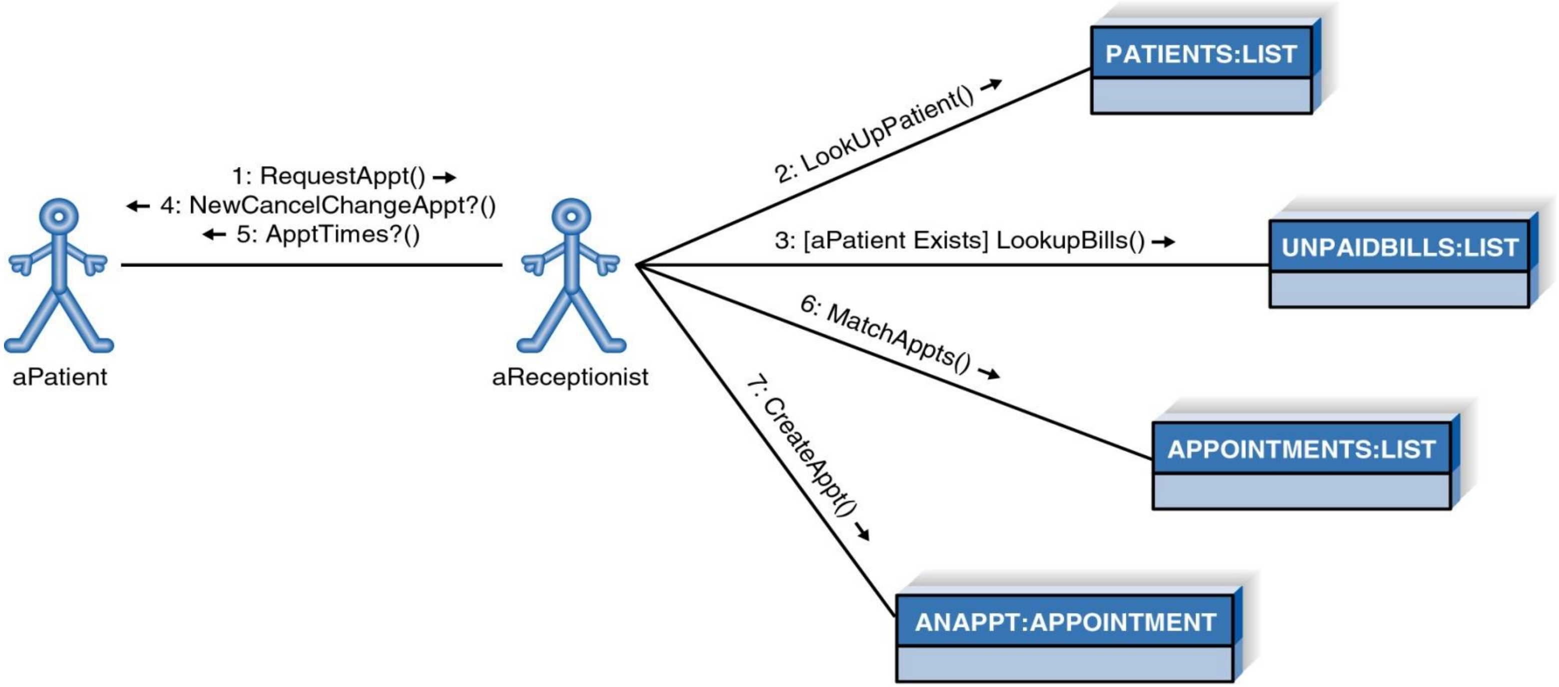
State of on object can be indicated
Initial state is indicated with <<create>>
If an object changes significantly during an interaction, you can add a new instance of the object to the diagram, draw a link between them and add a message with the stereotype <<become>>


1.1.3. GetBorrowersCheckedOutMedia(bID)
1.1.4. IsMediaOverdue(mID)
1.2. Display Invalid User Msg( )
1. Enter Borrower ID( )
1.1. CalcAmtCanBorrow(bID)
: UI
: Transaction
: Librarian
1.1.5. Amt Can Borrow
1.1.1. <<create>>
1.1.2. CalcBorrowerFines(bID)
: Fine

Change State of an Object



Example Collaboration Diagram



Dennis: SAD
Fig: 8-6 W-33 100% of size
Fine Line Illustrations (516) 501-0400

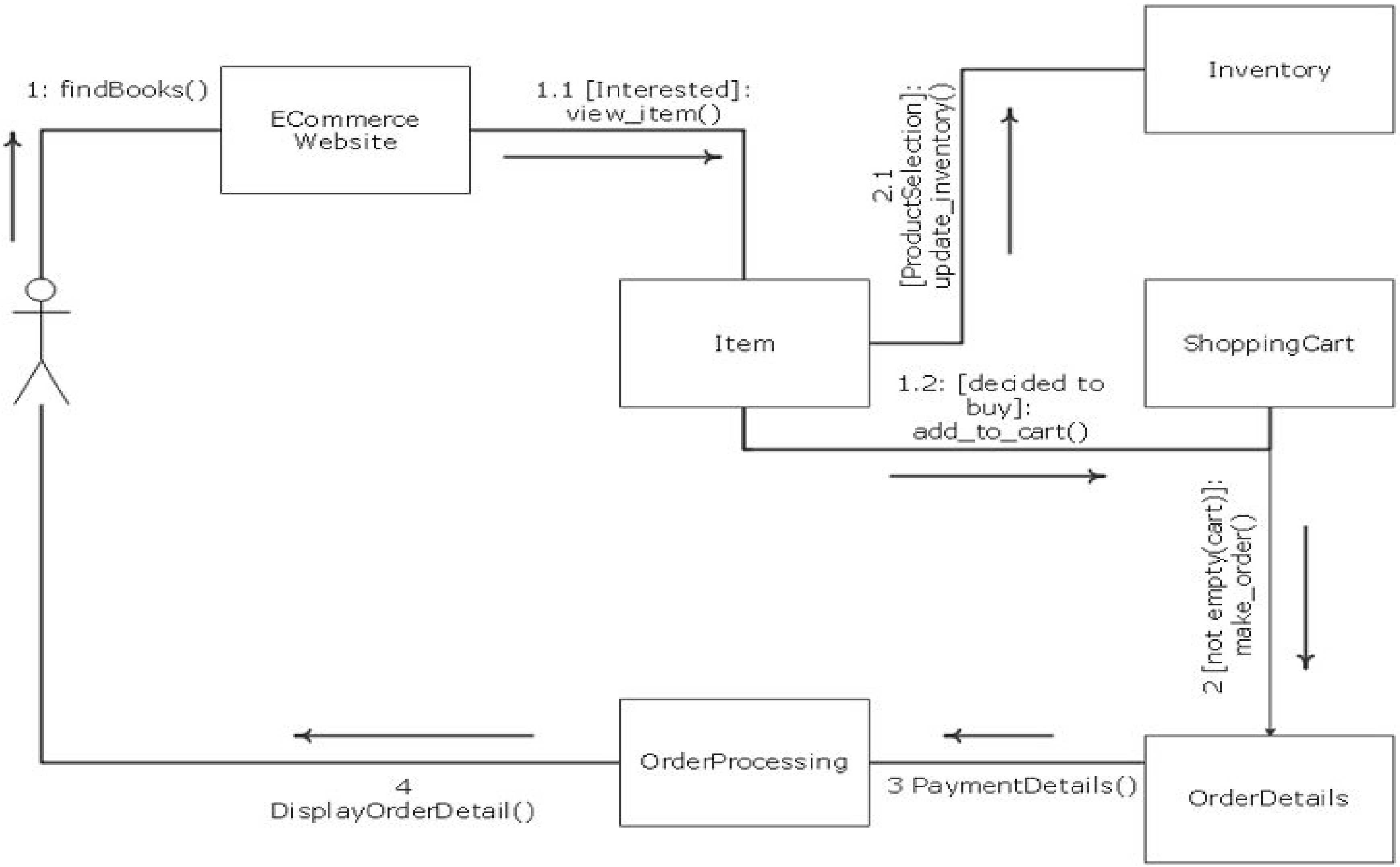
Application Scenarios of Collaboration Diagram

- Create a birds-eye view of a collection of objects collaborating, especially within a real-time system.
- Allocate capability to classes by exploring a system's behavioral attributes.
- Modelling collaborations, processes or hierarchical organization in the architecture of a system.
- Providing a description of objects operating together within an object-oriented framework.
- To display multiple alternate possibilities for the same use case.
- To illustrate forward and reverse engineering.
- Capturing information passing between objects.
- To visualize the complex logic of a system.

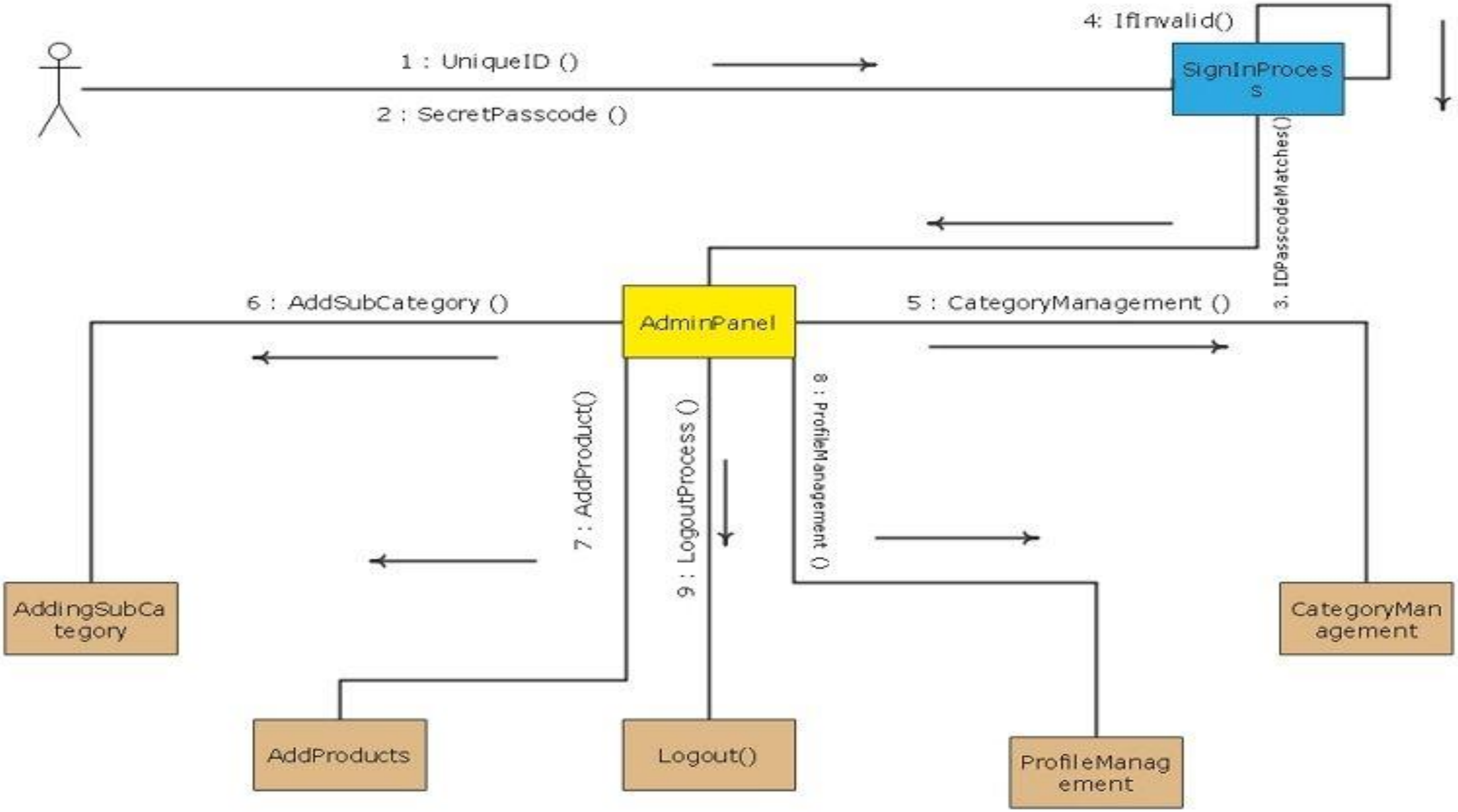
Benefits of Collaboration Diagram

- It reinforces the structural aspects of an interaction system which is how the lifeline is connected.
- Messages transmitted over sequencing are shown by hierarchical numeration of each message.
- It enables to focus on the structural elements and not on the flow of message as stated in sequence diagrams.

Collaboration Diagram
For Purchase Journey on Ecommerce Wesite


1: findBooks()
ECommerce Website
1.1 [Interested]: view_item()
Inventory
2.1 [ProductSelection]: update_inventory()
Item
ShoppingCart
1.2: [decided to buy]: add_to_cart()
2 [not empty(cart)]: make_order()
OrderProcessing
3 PaymentDetails()
OrderDetails
4 DisplayOrderDetail()

Collaboration Diagram For AdminPanel



Let’s refine the collaboration diagram to include the objects as specified:

Objects/Participants:

- User: Represents the person interacting with the ATM system.
- Bank Account: Represents the user’s bank account.
- Database: Represents the database system storing user account information.

Messages:

- User initiates a transaction request by interacting with the ATM interface.
- ATMController (not explicitly mentioned but implied): Receives transaction requests from the user.
- ATMController communicates with the Database to retrieve account information and verify user credentials.
- Database responds to the ATMController with account details and authentication status.
- ATMController instructs the BankAccount object to perform the requested transaction (e.g., withdrawal, balance inquiry).
- BankAccount updates its state based on the transaction outcome.
- Status updates or error messages are sent back and forth between the objects as needed.

- Association Links:
  - User is associated with the ATMController to initiate transactions.
  - ATMController is associated with the Database to retrieve and update account information.
  - ATMController is associated with BankAccount to perform transactions.
- Roles:
  - User: Initiates transactions by interacting with the ATM.
  - ATMController: Manages the overall ATM operations, including communication with external systems (e.g., Database) and handling user requests.
  - Database: Stores and manages user account information, providing authentication and authorization services.
  - BankAccount: Represents the user’s account and performs transaction-related operations.

- **Interactions**:
  - User inserts card and enters PIN.
  - ATMController sends user credentials to the Database for validation.
  - Database authenticates the user and retrieves account information.
  - ATMController presents transaction options to the user.
  - User selects a transaction (e.g., withdrawal).
  - ATMController communicates with the Database to update the account balance.
  - Database updates the BankAccount object with the transaction details.
  - Confirmation message is displayed to the user.
- **Constraints**:
  - Authentication must be successful before allowing any transaction.
  - Proper authorization checks must be performed before performing sensitive operations (e.g., withdrawals).
  - Error handling mechanisms should be in place for database communication failures or transaction errors.
  - Security measures should be implemented to protect sensitive user information during communication with the database.

Thank You