



# CS-3004

## SOFTWARE DESIGN AND ANALYSIS

RUBAB JAFFAR

[RUBAB.JAFFAR@NU.EDU.PK](mailto:RUBAB.JAFFAR@NU.EDU.PK)

# Interaction diagrams

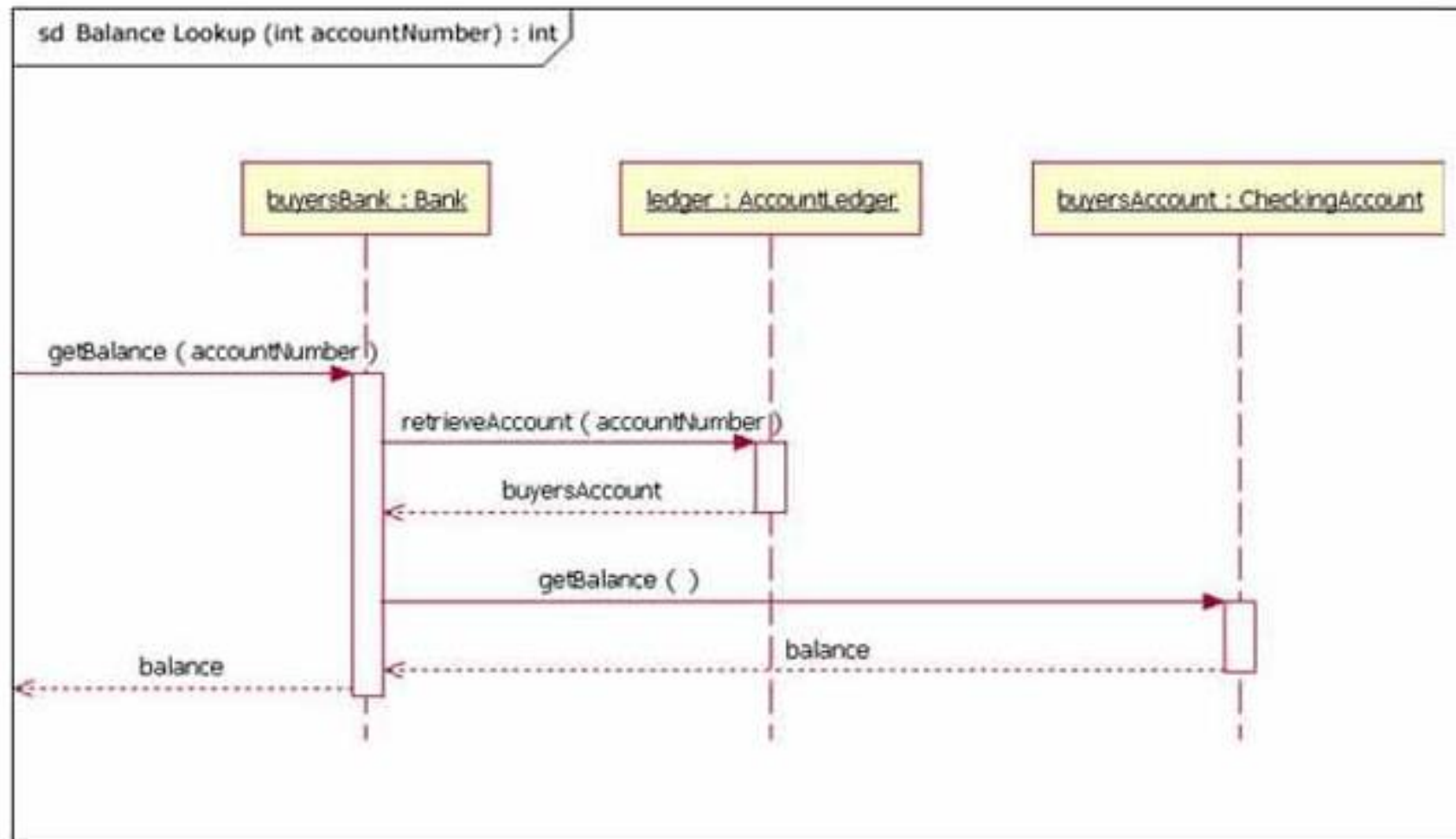
# TODAY'S OUTLINE

- Interaction Diagrams
  - Sequence Diagram
  - Collaboration Diagram
- Sequence Diagram
- Sequence Diagram Notations
- Sequence Diagram Example
- Sequence Diagram Advanced Notations
- Sequence Diagram Example

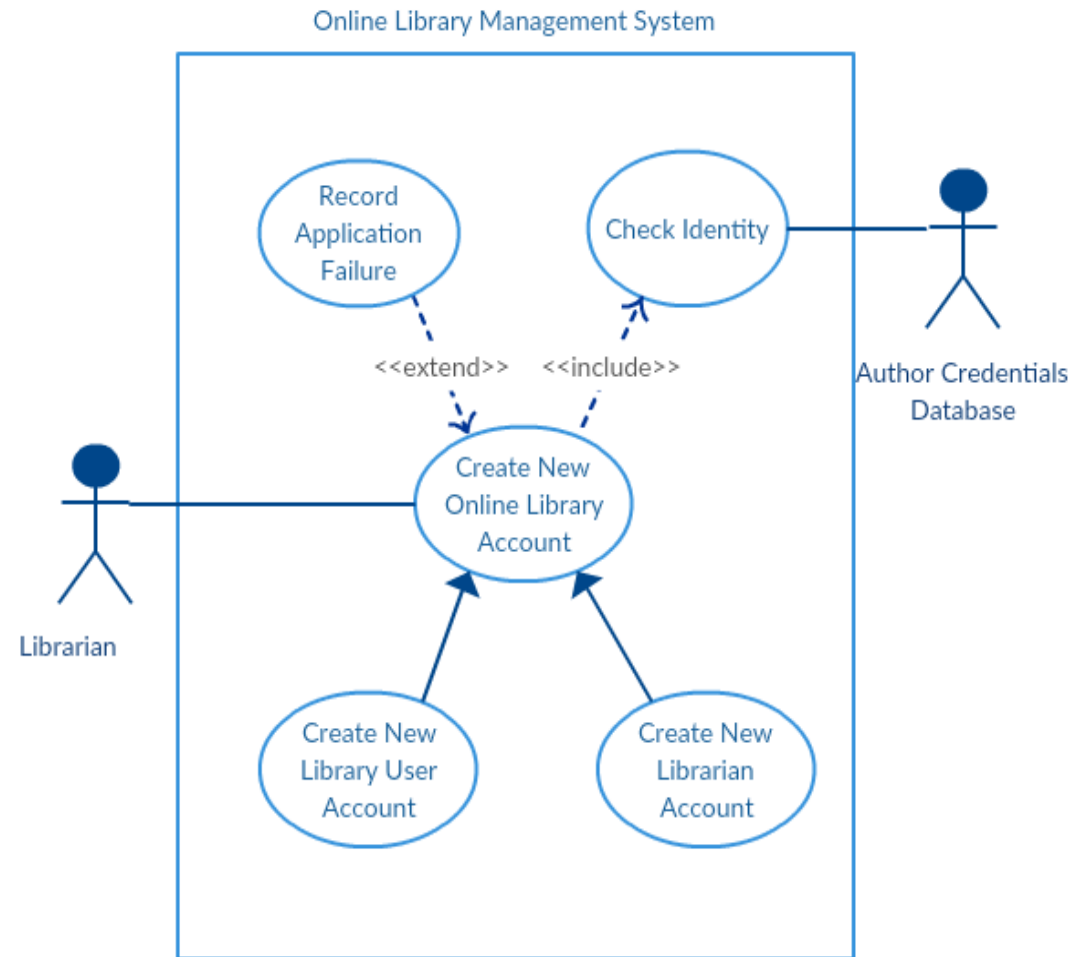
## EXAMPLE: BALANCE LOOKUP USE CASE

- Customer asks the bank for his account balance.
- Bank verifies his account number from the account ledger.
- Bank then sends message to checkaccount for getting the balance.
- The bank informs the customer about balance.

# SEQUENCE DIAGRAM EXAMPLE



# EXAMPLE: SEQUENCE DIAGRAM FROM USE CASES



# USE CASE DESCRIPTION

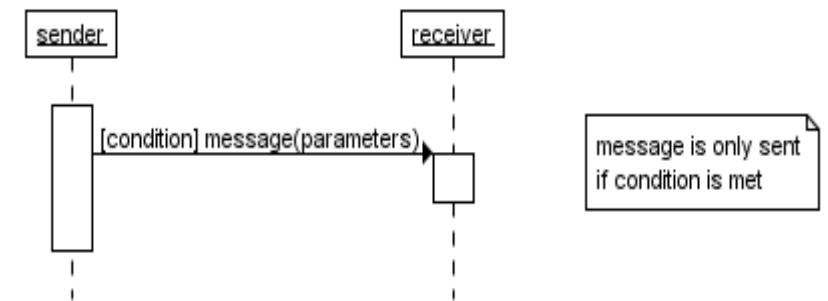
- Here are the steps that occur in the use case named 'Create New Library User Account'.
  - The librarian request the system to create a new online library account
  - The librarian then selects the library user account type
  - The librarian enters the user's details
  - The user's details are checked using the user Credentials Database
  - The new library user account is created
  - A summary of the of the new account's details are then emailed to the user

# EXAMPLE: 'CREATE NEW USER ACCOUNT' --SEQUENCE DIAGRAM.

- Before drawing the sequence diagram, it's necessary to identify the objects or actors that would be involved in creating a new user account. These would be;
- Librarian
- Online Library Management system
- User credentials database
- Email system

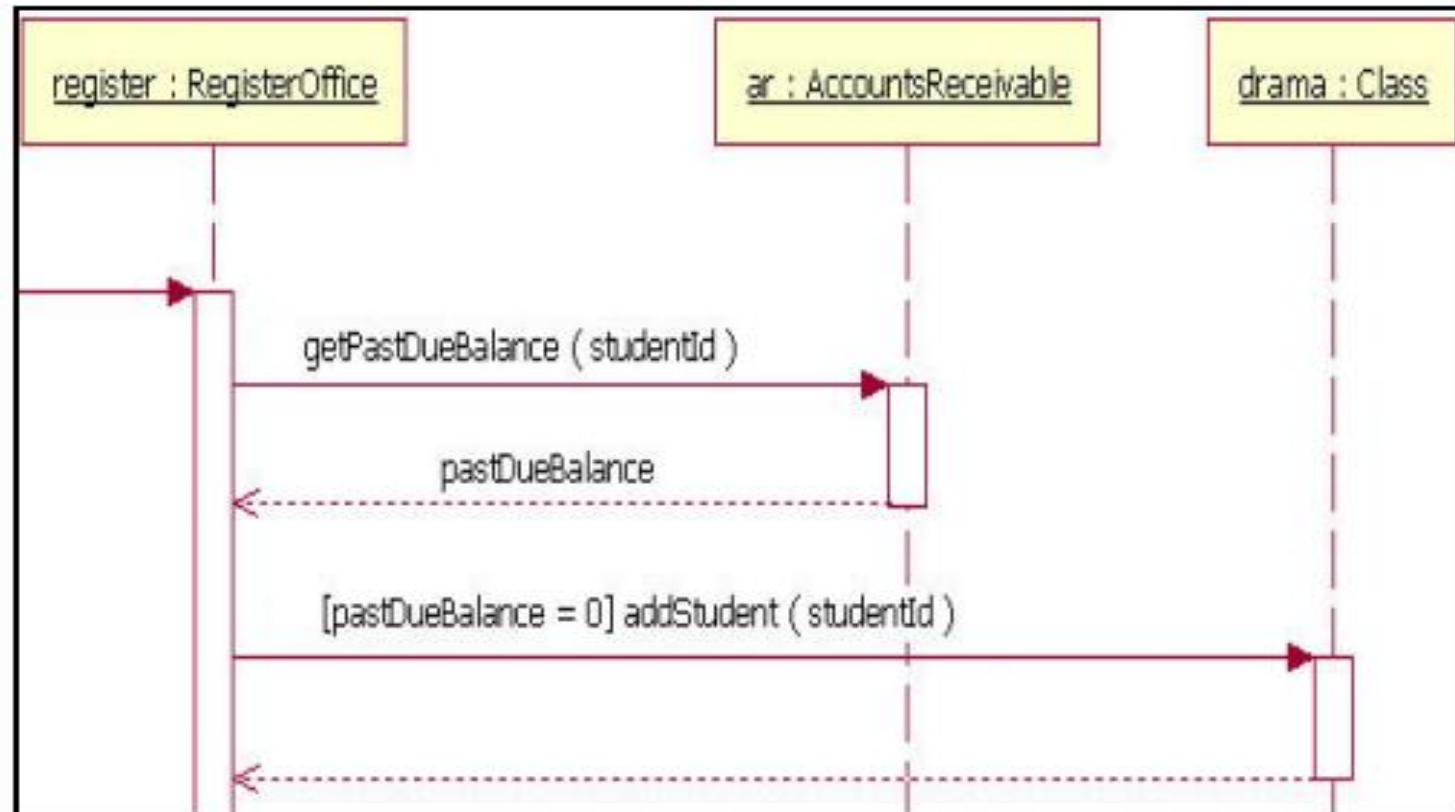
# GUARDS

- There will be times when a condition must be met for a message to be sent to the object. i.e Conditional interaction
- There will be certain prerequisite for communication or a message to be sent to the sender.
- These conditions are attributed as “Guard” in sequence diagram, a guard behaves like “if statements” in the sequence diagram.
- They are used to control the flow of the messages between objects.



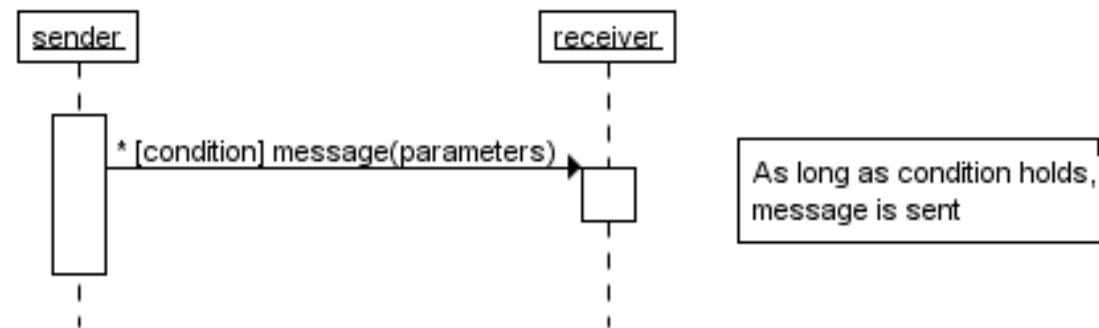


# GUARDS



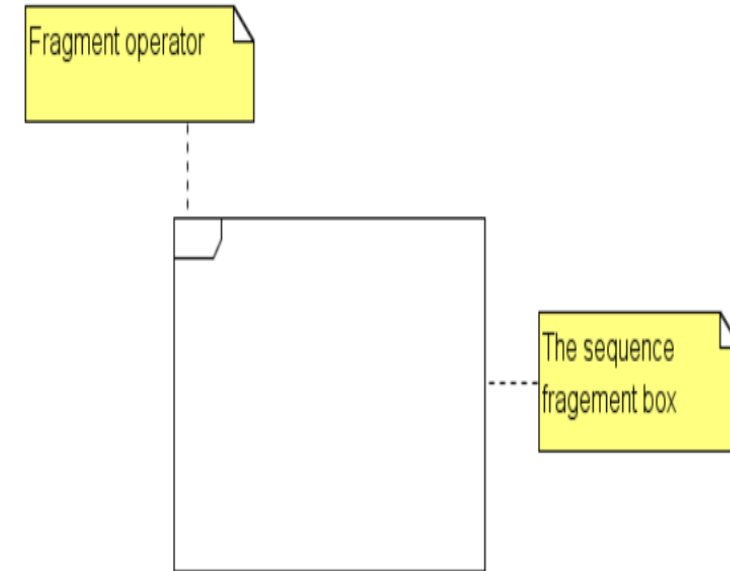
# REPEATED INTERACTION

- When a message is prefixed with an asterisk (the '\*'-symbol), it means that the message is sent repeatedly. A guard indicates the condition that determines whether or not the message should be sent (again). As long as the condition holds, the message is repeated.



# FRAGMENTS

- Sequence diagrams can be broken up into chunks called fragments or combined fragments.
- **Manage complex interactions with sequence fragments**
- It is used to show complex interactions such as alternative flows and loops in a more structured way. On the top left corner of the fragment sits an operator. This – the fragment operator – specifies what sort of a fragment it is.
  - Fragment types: ref, loop, break, alt, opt, parallel
- Sequence fragments make it easier to create and maintain accurate sequence diagrams



# FRAME

- If -> (opt) [condition]
- if/else -> (alt) [condition],  
separated by horizontal dashed line
- loop -> (loop) [condition or  
items to loop over]

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.

# CONDITIONAL BEHAVIOR-OPTION

- The option combination fragment is used to model a sequence that, given a certain condition, will occur; otherwise, the sequence does not occur.
- An option is used to model a simple "if then" statement
  - Example, if there are fewer than five donuts on the shelf, then make two dozen more donuts.

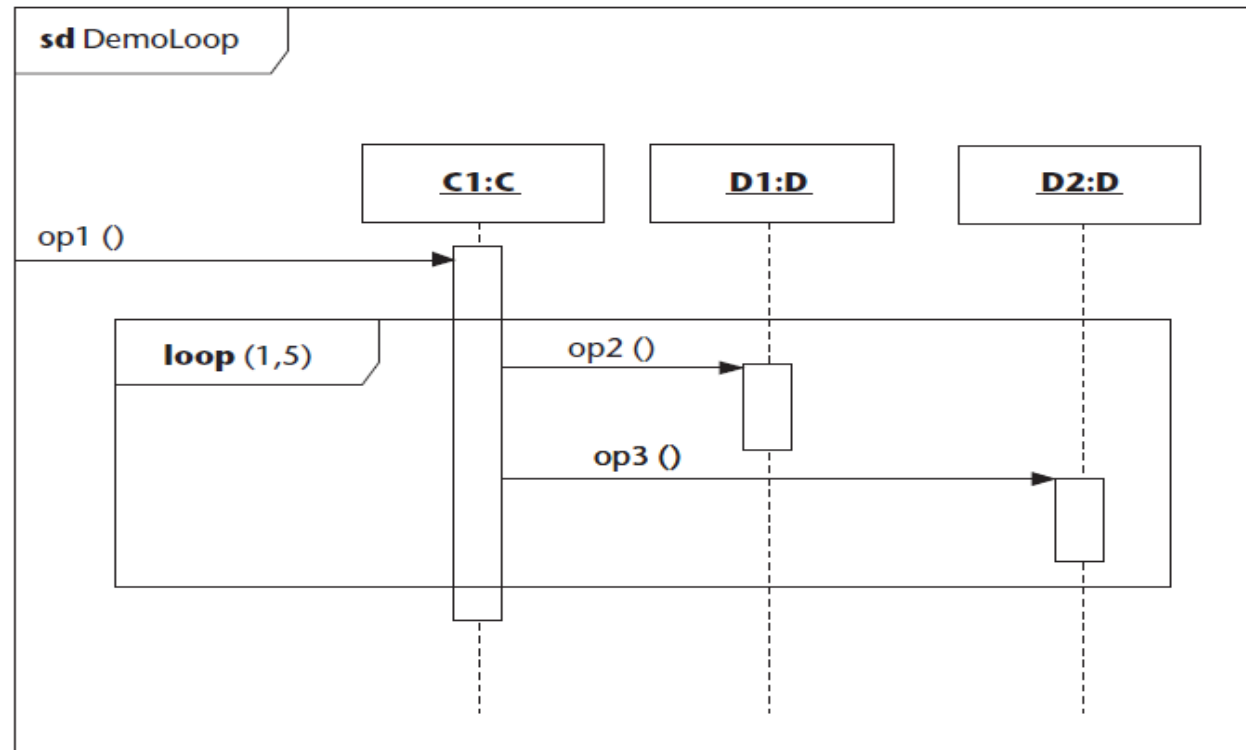
# CONDITIONAL BEHAVIOR-ALT

- When showing conditional behavior, the interaction operator keyword **alt** is put in the pentagram, the fragment is partitioned horizontally with a dashed line separator, and constraints are shown in square brackets .
- At most one of the alternatives occurs;

# LOOPS

- A repetition or loop within a sequence diagram is depicted as a frame.
- In the frame's name box the text "loop" is placed.
- Loops are designated by placing the interaction operator keyword loop in the pentagram. Textual syntax of the loop operand is “loop [ ‘( <minint> [,<maxint> ] ’ )” .
- Inside the frame's content area the loop's guard is placed towards the top left corner, on top of a lifeline.

# A SIMPLE EXAMPLE OF TWO OPERATIONS BEING REPEATED FIVE TIMES.

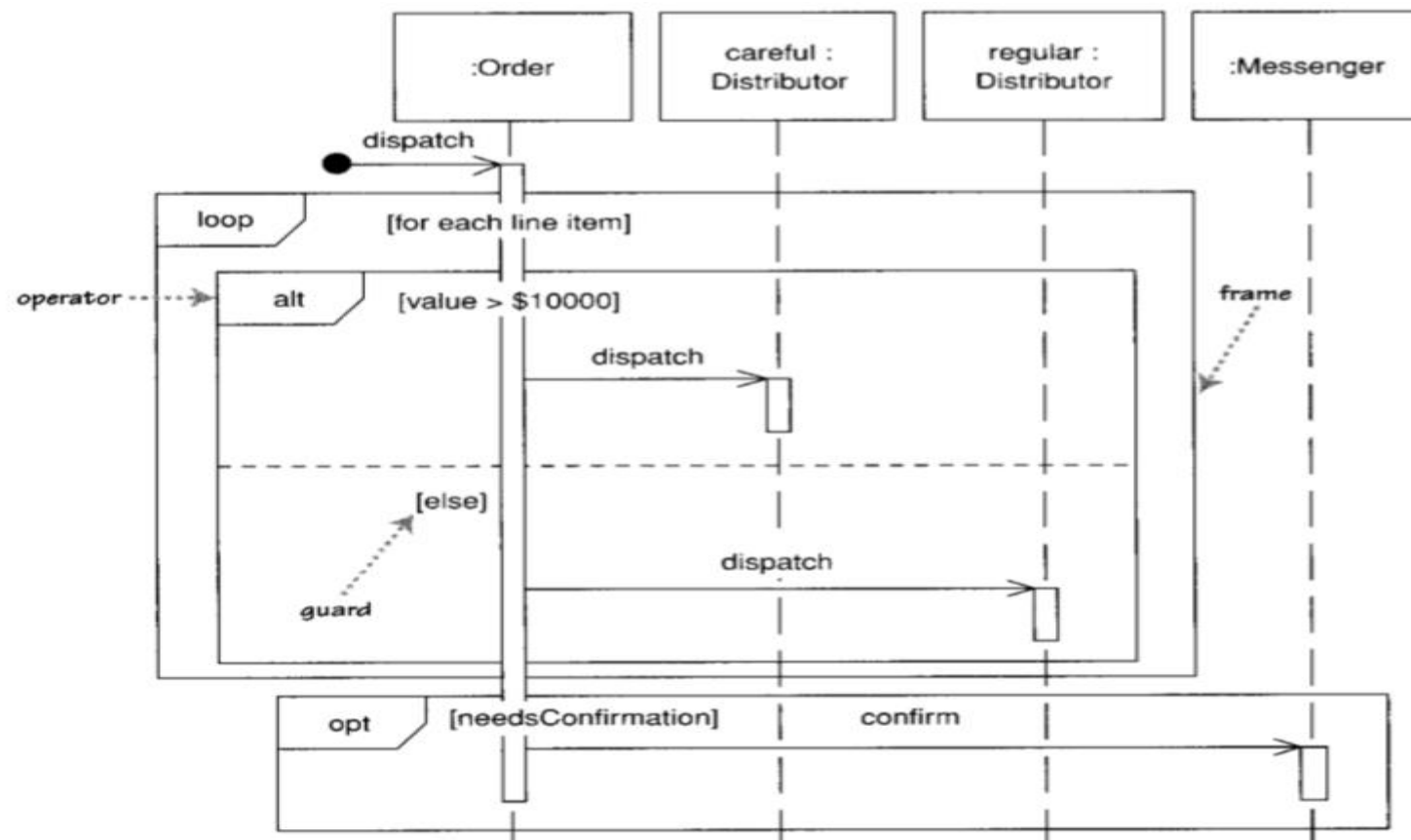


**Figure 5.31** Iteration expressed with a loop operand.



## EXAMPLE

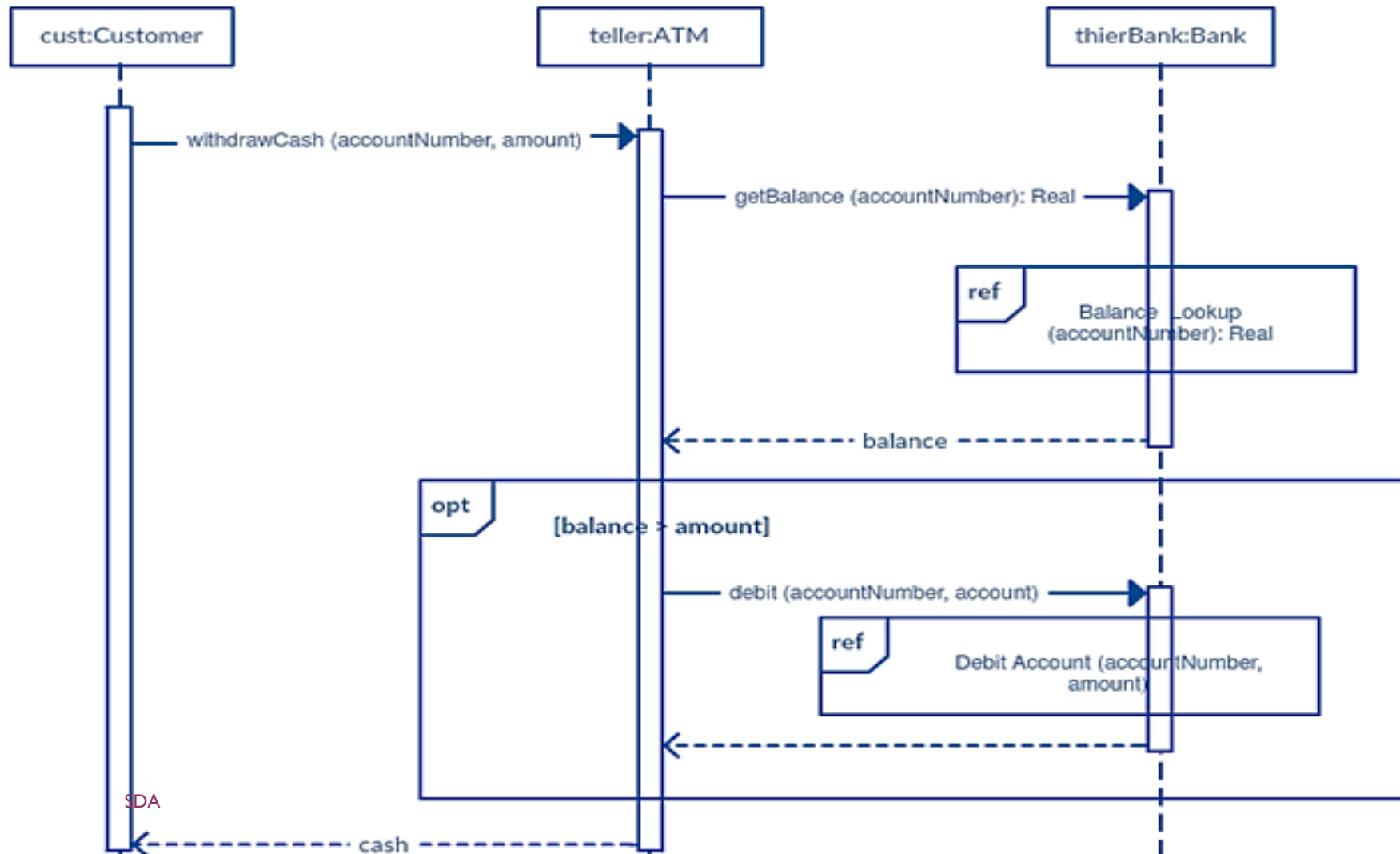
- When an **order** is received for dispatching, **each item in the order is checked for its value**. If the value is greater than 10000, it is sent to express **delivery dispatcher** other wise it is sent to **regular dispatcher**.
- If the **customer** has asked for confirmation, then **notification department** is asked to send confirmation.



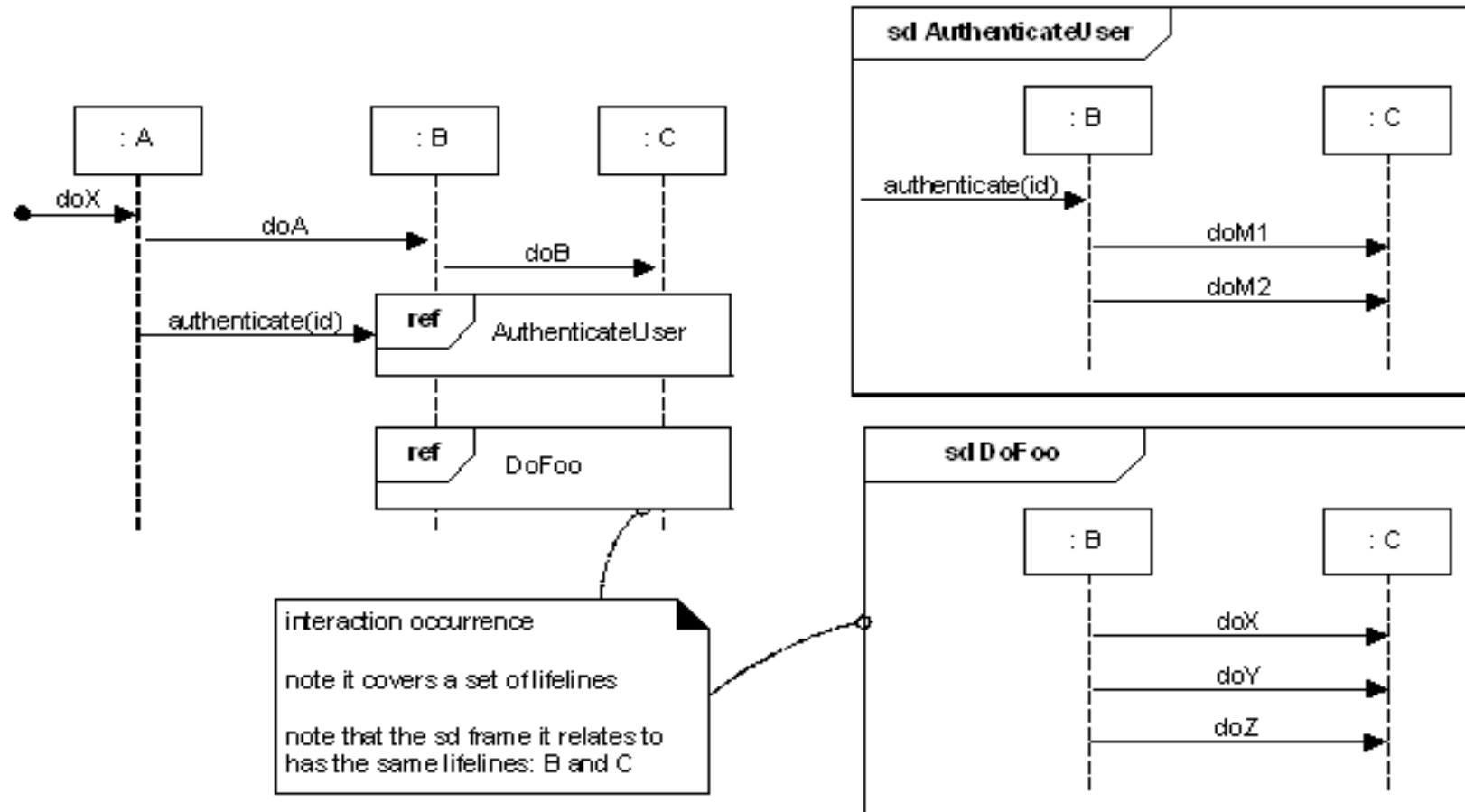
# REFERENCE FRAGMENT

- You can use the **ref** fragment to manage the size of large sequence diagrams. It allows you to reuse part of one sequence diagram in another, or in other words, you can reference part of a diagram in another diagram using the **ref fragment**.
- To specify the reference fragment, you have to mention 'ref' in the name box of the frame and the name of the sequence diagram that is being referred to inside the frame.

# REFERENCE FRAGMENT- EXAMPLE



# REFERENCE FRAGMENT- EXAMPLE



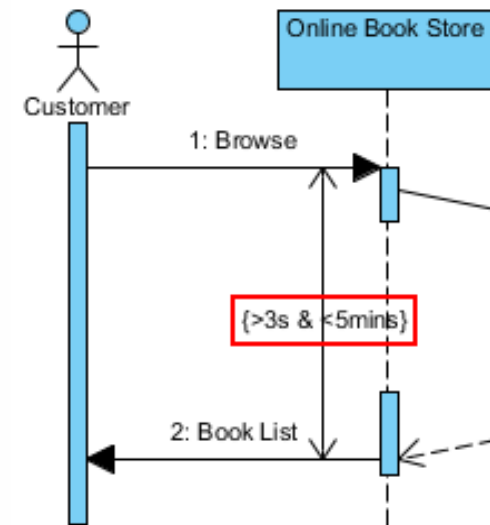
# COMMON OPERATORS FOR INTERACTION FRAMES

Operator	Meaning
alt	<b>Alternative multiple fragments:</b> only the one whose condition is true will execute.
opt	<b>Optional:</b> the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	<b>Parallel:</b> each fragment is run in parallel.
loop	<b>Loop:</b> the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	<b>Critical region:</b> the fragment can have only one thread executing it at once.
neg	<b>Negative:</b> the fragment shows an invalid interaction.
ref	<b>Reference:</b> 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	<b>Sequence diagram:</b> used to surround an entire sequence diagram.

# SPECIFYING TIMING REQUIREMENTS

- When modeling a real-time system, or even a time-bound business process, it can be important to consider the length of time it takes to perform actions.
- Assume you need to specify the time limit between Browse message and Book List message, you therefore, have to add duration constraint between them.
- For example, it should take more than 3 seconds but less than 5 minutes.

You can enter  $> 3s$  &  $< 5mins$ .



# HOW TO PRODUCE SEQUENCE DIAGRAMS

1. Decide on Context:
  - a) Identify behavior (or use case) to be specified
2. Identify structural elements:
  - a) Model objects (classes)
  - b) Model lifelines
  - c) Model activations
  - d) Model messages
  - e) Model Timing constraints
3. Elaborate as required



# WHY NOT JUST CODE IT?

- Sequence diagrams can be somewhat close to the code level. So why not just code up 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 all code is drawn on diagram)
- 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)

# SEQUENCE DIAGRAMS AND USE CASES SYSTEM

## SEQUENCE DIAGRAM

- System sequence diagrams are actually a sub-type of sequence diagrams.
- Sequence diagrams show the progression of events over a certain amount of time, while system sequence diagrams go a step further and present sequences for specific use cases.

# SEQUENCE DIAGRAMS AND USE CASES SYSTEM

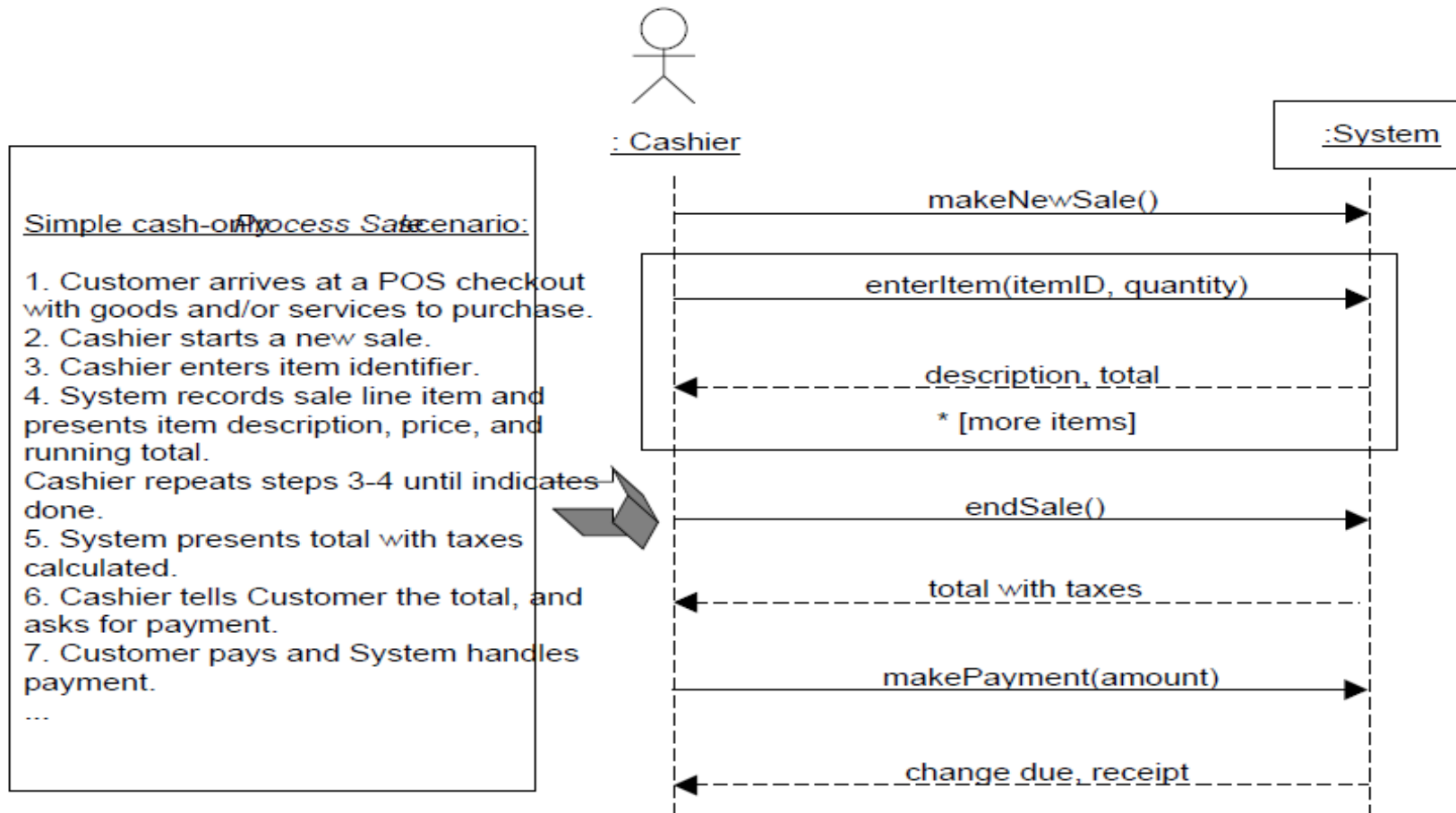
## SEQUENCE DIAGRAM

- SSD: The elements participating (exchanging messages) in a system sequence diagram are *Actors* and *Systems*. The messages exchanged by these elements could be any type depending on the systems (from web service calls to data input from a human).
- SD: The elements participating in a sequence diagram are objects (instances of various classes). The messages exchanged by these elements are method invocations.

# SYSTEM SEQUENCE DIAGRAM (SSD)

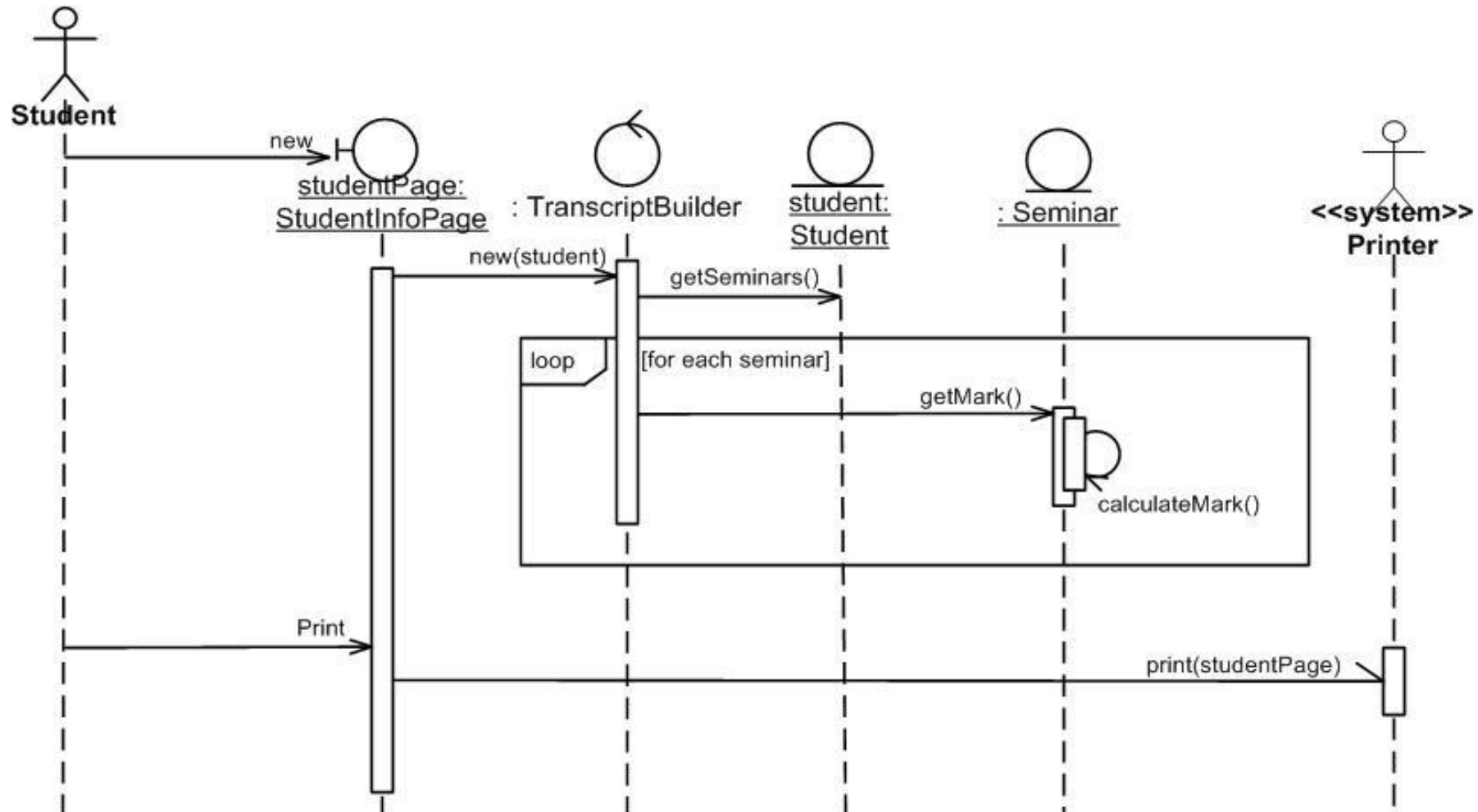
- A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and inter-system events.
- All systems are treated as a black box; the emphasis of the diagram is events that cross the system boundary from actors to systems.
- An SSD should be done for the main success scenario of the use case, and frequent or complex alternative scenarios.

# SYSTEM SEQUENCE DIAGRAM (SSD)



SSDs are derived from use cases.

# SEQUENCE DIAGRAM FOR STUDENTS SEMINAR TRANSCRIPT



## YOUR TURN

### **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.

## YOUR TURN (SD BY ECB CLASSES)

- **Customer requests reservation clerk for the room reservation.**
- **Reservation clerk checks the availability of hotel rooms through hotel control system.**
- **Customer can also request discount.**
- **Reservation clerk provide information about total cost by checking the rate and bill from the hotel control system.**
- **Customer confirms the reservation of room and room is reserved.**
- **Customer is charged by the hotel control system through the credit card.**

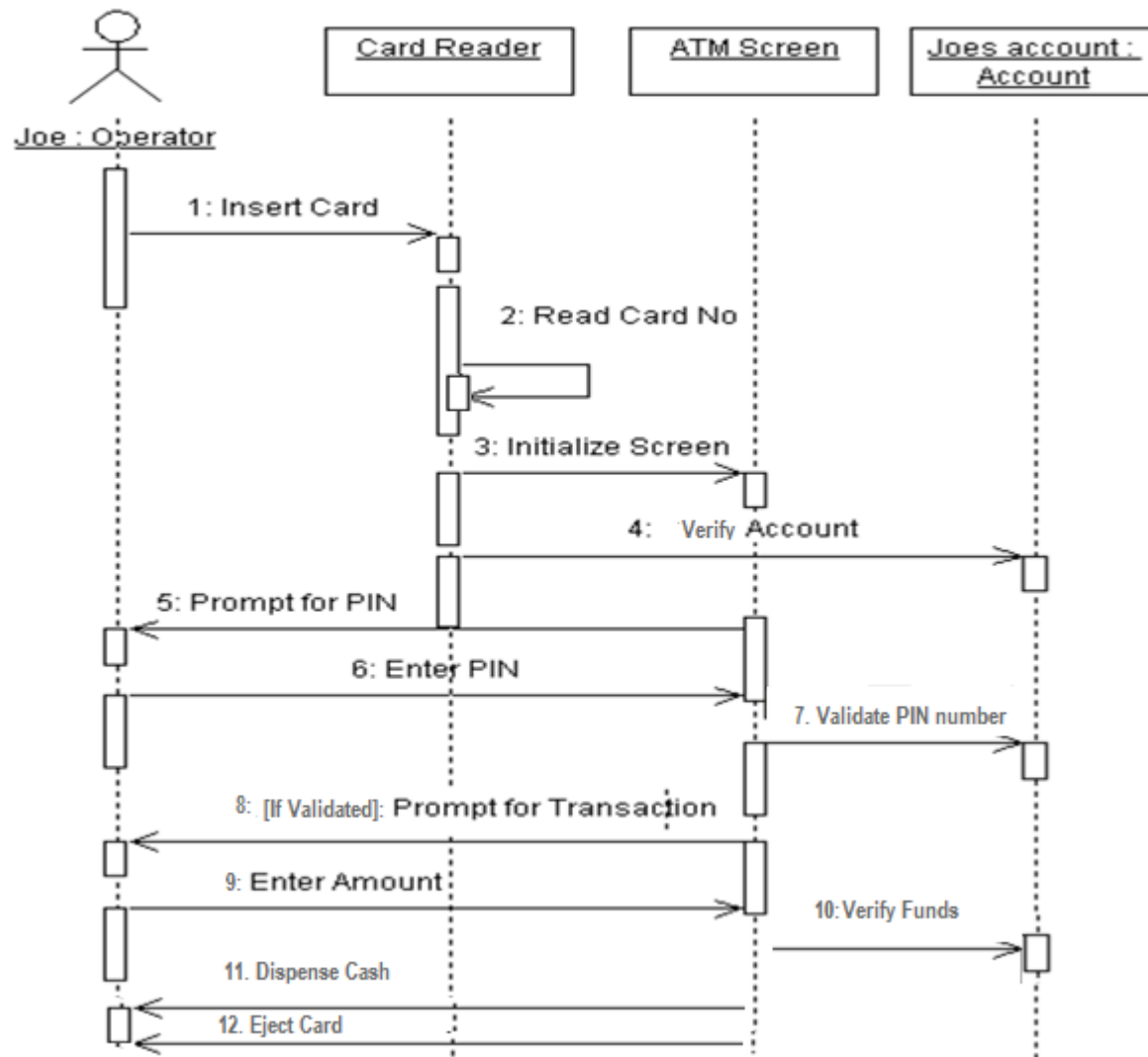


## YOUR TURN

- Check in Process Use Case
- Person arrives at desk, gives name.
- Receptionist gets map of available Rooms, chooses one.
- Receptionist gets details of user
- Receptionist enters Person's details on Room Management system against chosen room for room confirmation
- Receptionist gives key to Person.

## YOUR TURN: 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 ATM verifies the card against account and prompts Joe for his PIN.
  - Joe enters PIN.
  - Joe's PIN is validated and the ATM prompts him for a transaction
  - Joe selects Withdraw Money
  - The ATM prompts Joe for an amount.
  - Joe enters \$ 20.
  - The ATM verifies that Joe's account has sufficient funds and subtracts \$ 20 from his account.
  - The ATM dispenses \$ 20 and ejects Joe's card



# Your Turn

- 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.

# TODAY'S OUTLINE

- Collaboration Diagram Semantics
- Collaboration Diagram Notation
- Collaboration Diagram Examples
- Collaboration Diagram Issues

# COLLABORATION DIAGRAM SEMANTICS

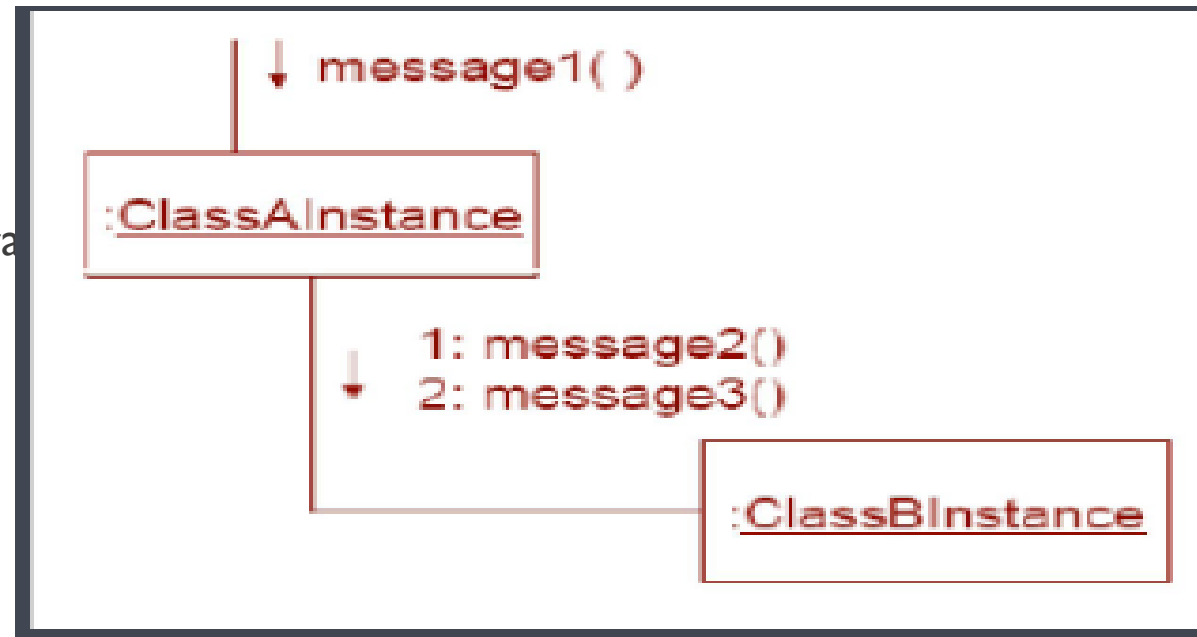
- Member of the Behavioral Group of diagrams
- Collaboration Diagrams captures dynamic behavior of the objects in the system.
- They are very useful for visualizing the relationship between objects collaborating to perform a particular task.

# COLLABORATION DIAGRAM

- Represents a Collaboration and Interaction
- **Collaboration** set of objects and their interactions in a specific context
- **Interaction** set of messages exchanged in a collaboration to produce a desired result
- Their purpose is to:
  - Model flow of control
  - Illustrate coordination of object structure and control

# WHAT IT REPRESENTS?

- Collaboration Diagram

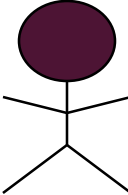







# COLLABORATION DIAGRAM ELEMENTS

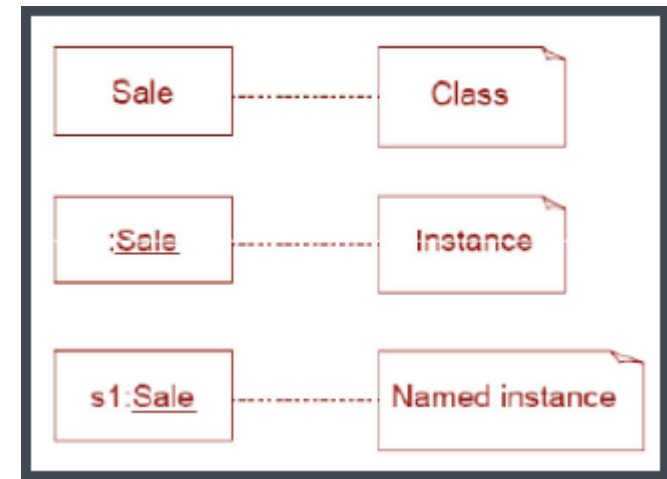
- There are three primary elements of a collaboration diagram:
  - Objects
  - Links
  - Messages

# Collaboration Diagram Syntax

AN ACTOR	
AN OBJECT	
AN ASSOCIATION	
A MESSAGE	

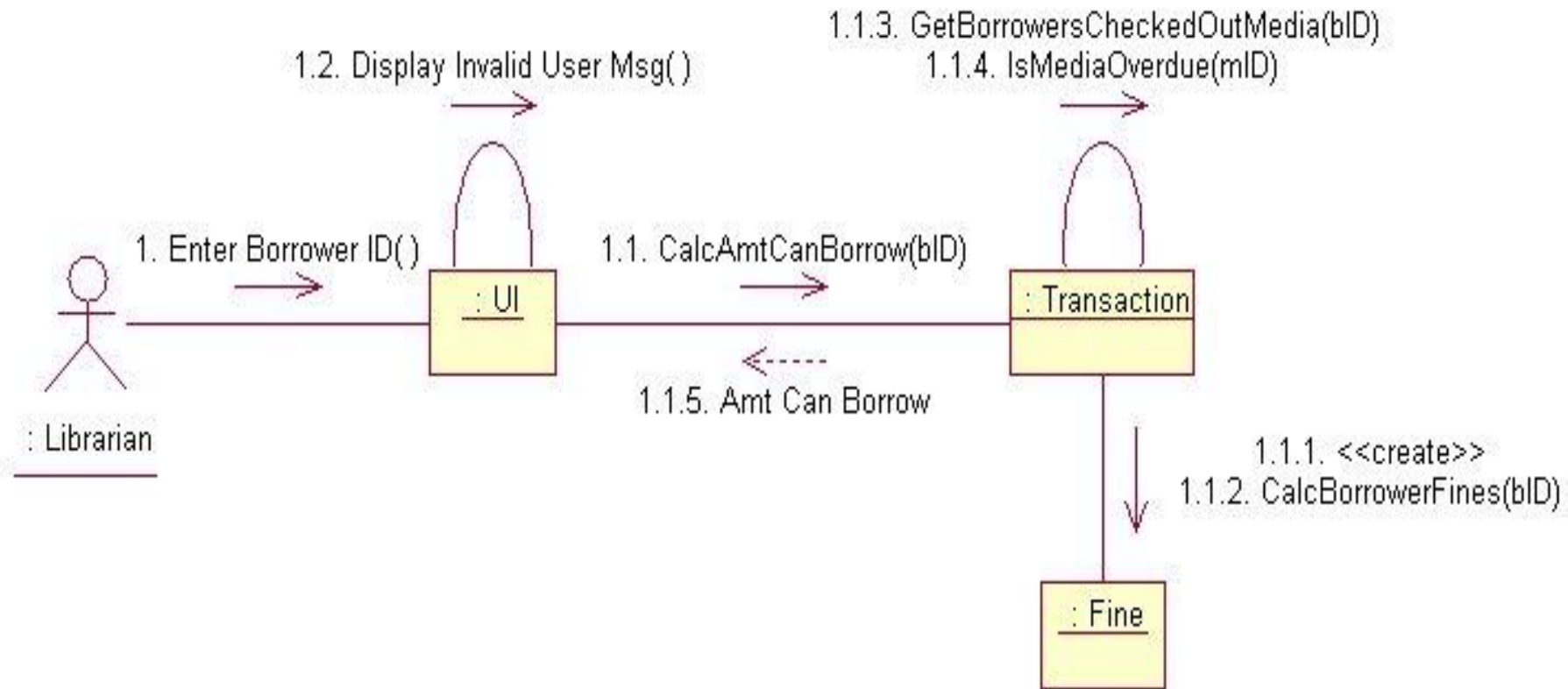
# NOTATIONS USED FOR COLLABORATION DIAGRAMS-**OBJECTS**

- **Objects** rectangles containing the object signature
  - **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



# NOTATIONS USED FOR COLLABORATION DIAGRAMS-OBJECTS

- 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



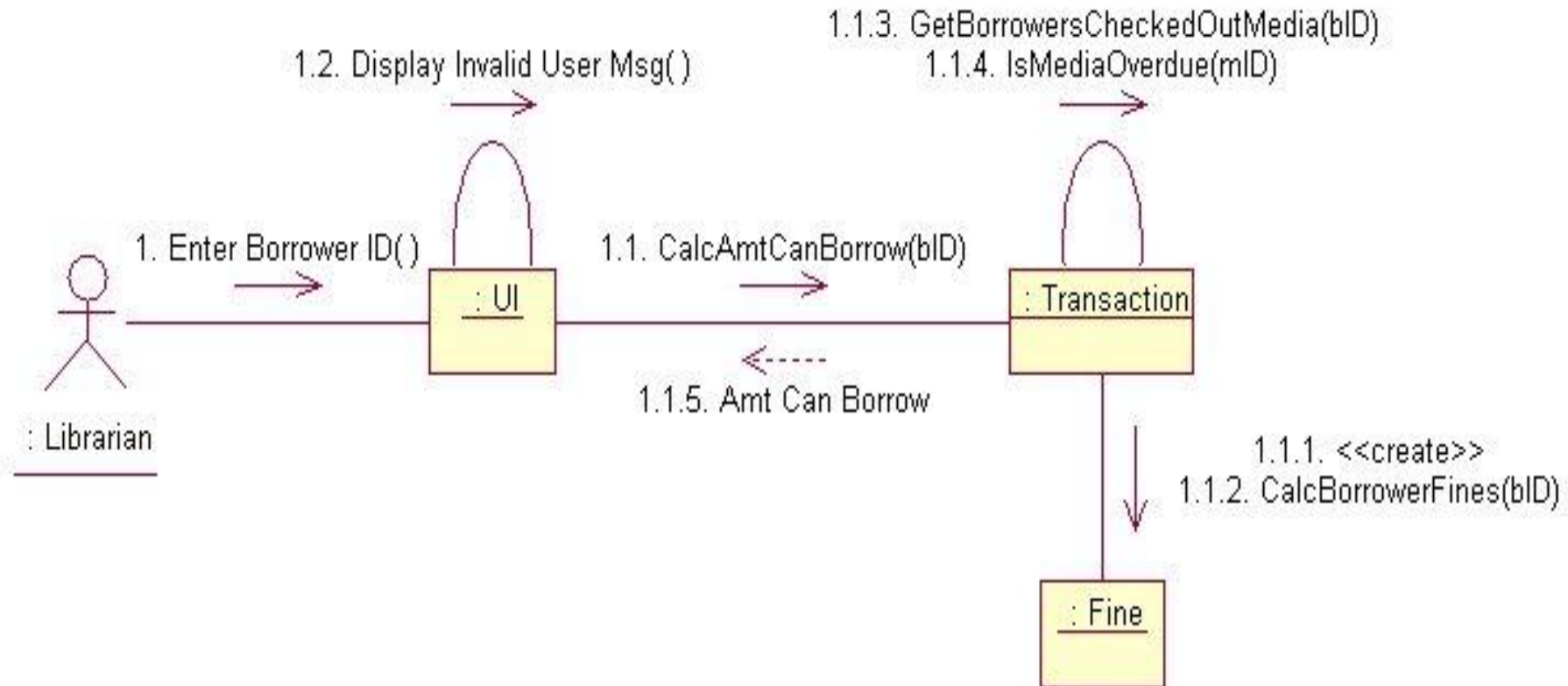
TRANSACTION OBJECT ACTS AS A SUPPLIER TO THE UI (USER INTERFACE) CLIENT OBJECT.

IN TURN, THE FINE OBJECT IS A SUPPLIER TO THE TRANSACTION CLIENT OBJECT.

# NOTATIONS USED FOR COLLABORATION DIAGRAMS-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

# Notations used for Collaboration Diagrams-Links



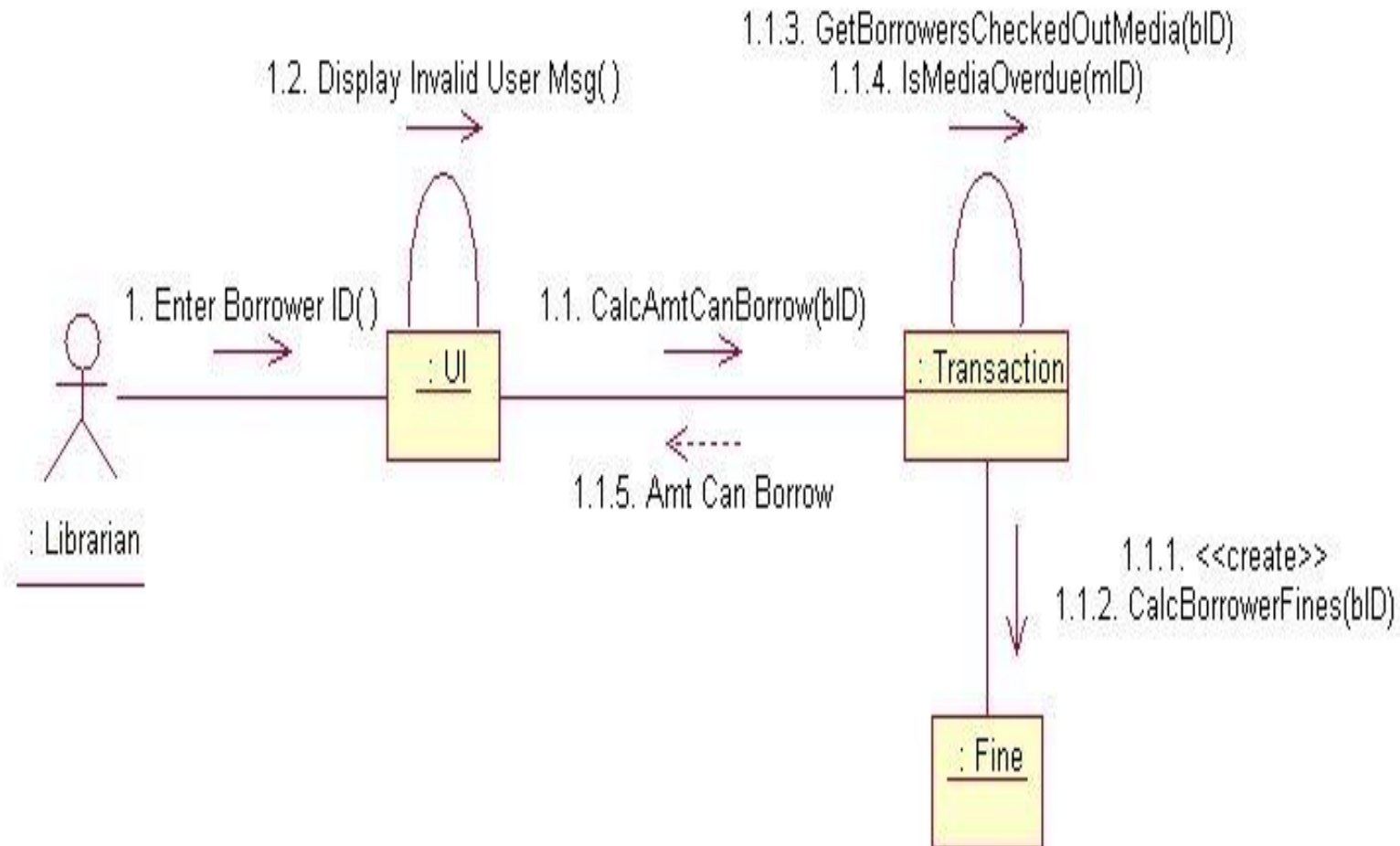
The visual representation of a link is a straight line between two objects. If an object sends messages to itself, the link carrying these messages is represented as a loop icon. This loop can be seen on both the UI object and the Transaction object.

# NOTATIONS USED FOR COLLABORATION DIAGRAMS- MESSAGES

- An interaction between objects is implemented 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**.



# NOTATIONS USED FOR COLLABORATION DIAGRAMS-MESSAGES



# ITERATING MESSAGES

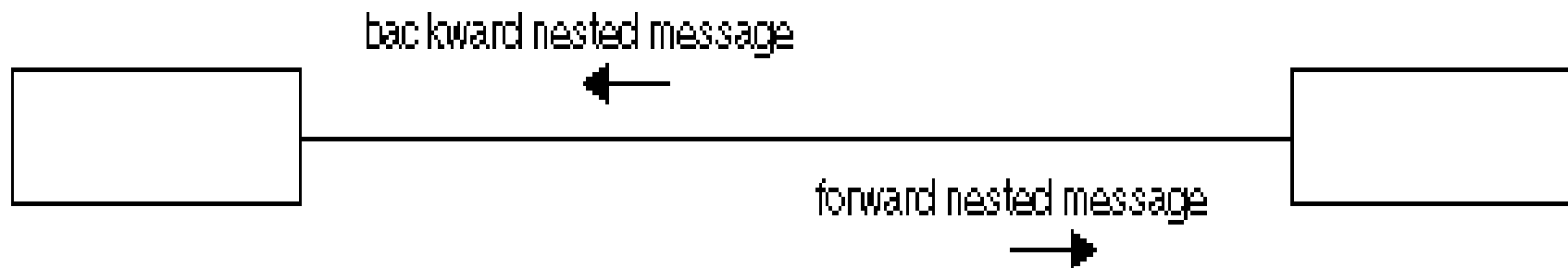
- Collaboration diagrams use syntax similar to sequence diagrams to indicate that either a message iterates (is run multiple times) or is run conditionally
  - An asterisk (\*) indicates that a message runs more than once
  - Or the number of times a message is repeated can be shown by numbers (for example,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

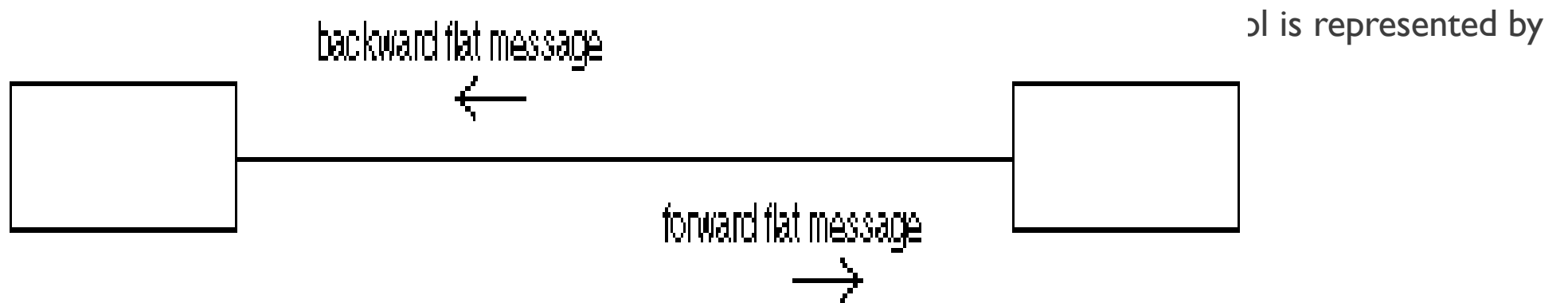
# NESTED MESSAGE

- The nested message represents a procedure call or other nested flow of control. The nested sequence is completed before the outer level sequence resumes. The nested message symbol is represented by a filled solid arrowhead.



# FLAT MESSAGE

- The flat message is represented by a stick arrow



# 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 an 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>>`

# CHANGE STATE OF AN OBJECT





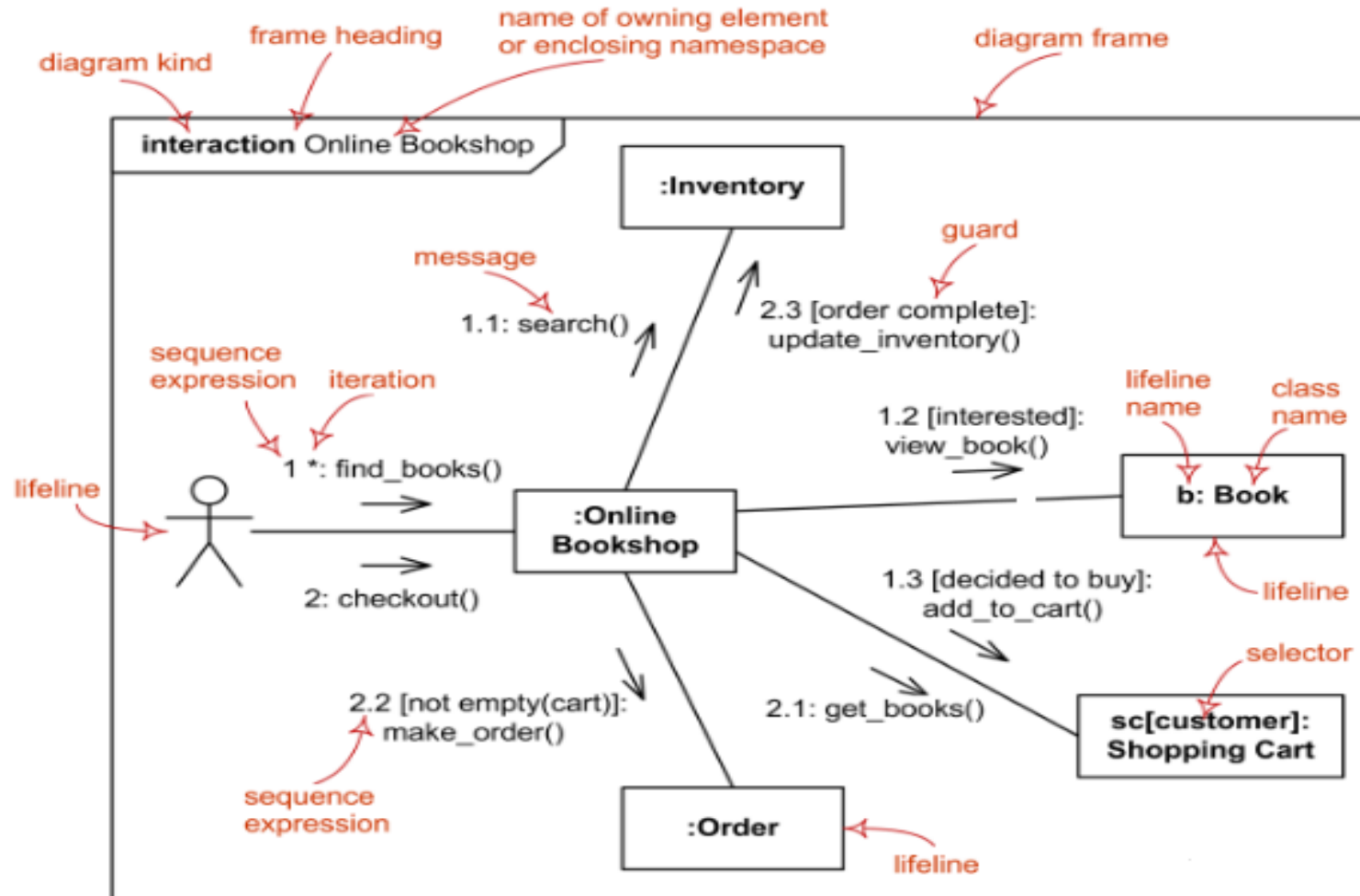
# STEPS TO CREATING A COLLABORATION DIAGRAM

1. Determine the scope of the diagram- the use case it relates to.
2. Place the objects that participate in the collaboration on the diagram
  - Remember to place the most important objects towards the center of the diagram.
3. 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.

## STEPS TO CREATING A COLLABORATION DIAGRAM

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

# ONLINE BOOK PURCHASE COLLABORATION DIAGRAM



# COLLABORATION VS SEQUENCE DIAGRAM

- In reality, sequence diagrams and collaboration diagrams show the same information, but just present it differently
- Not used as often as sequence diagrams but are closely related
- *Sequence Diagrams* are arranged according to Time.
- *Collaboration Diagrams* represent the structural organization of object.
- [Both sequence and collaboration diagrams are called **interaction** diagrams]

# COLLABORATION VS SEQUENCE DIAGRAM

- 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.
- The difference between sequence diagrams and collaboration diagrams is that collaboration diagrams emphasize more the structure than the sequence of interactions.

# COLLABORATION VS SEQUENCE DIAGRAM

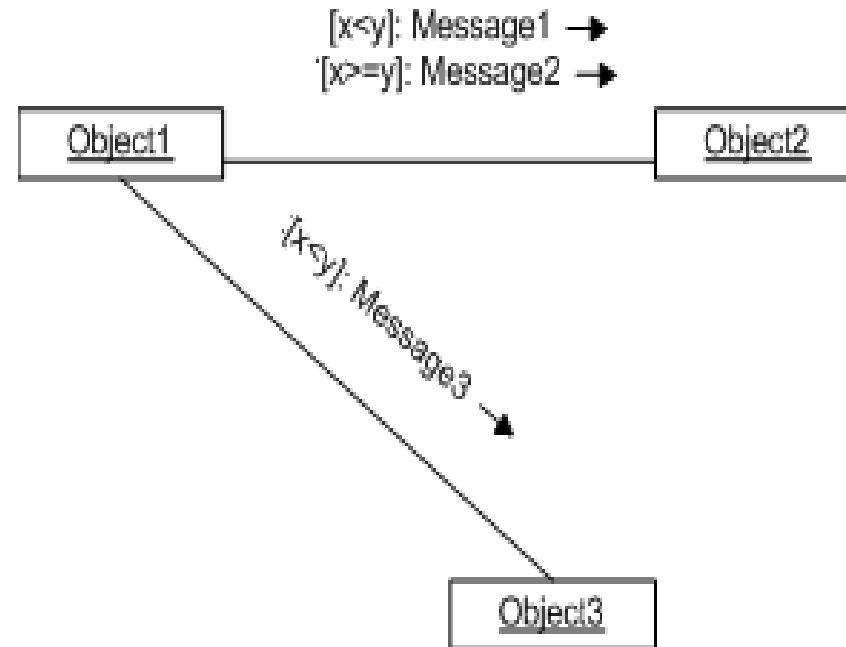
- Collaboration diagram illustrates object interactions in a graph or network format, in which objects can be placed anywhere on the diagram. It demonstrates how objects are statically connected.
- Sequence diagram illustrates interaction in a kind of fence format, in which each new object is added to the right. It generally shows the sequence of events that occur.

# WHY COLLABORATION DIAGRAMS?

- Collaboration diagrams offer a better view of a scenario than a Sequence diagram when the modeler is trying to understand all of the effects on a given object and are therefore good for procedural design.
- A distinguishing feature of a Collaboration diagram is that it shows the objects and their association with other objects in the system apart from how they interact with each other. The association between objects is not represented in a Sequence diagram.

# IF-ELSE

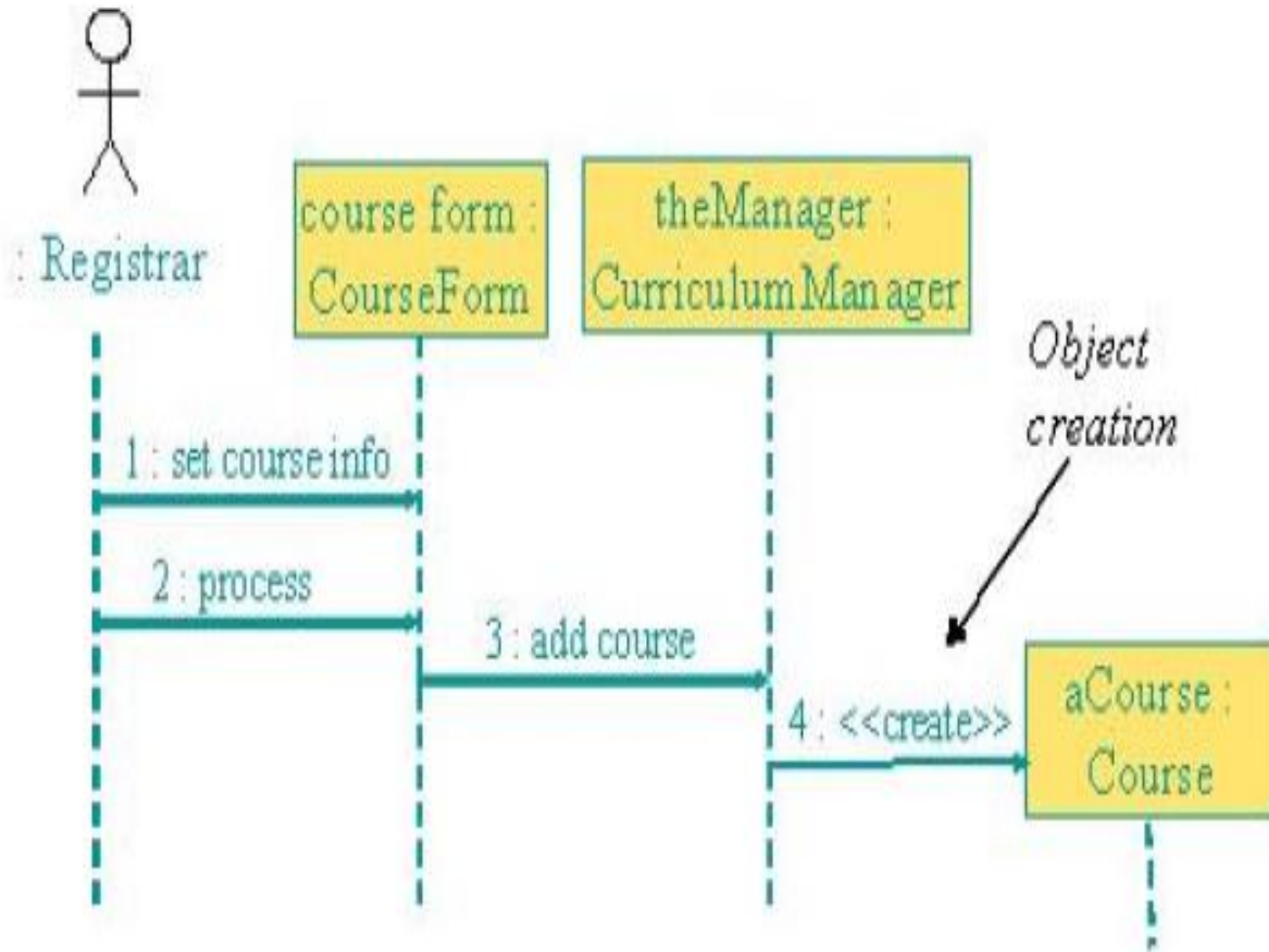
```
if (x<y)
{
    object2.message1();
    object3.message3();
}
else
{
    object2.message2();
}
```



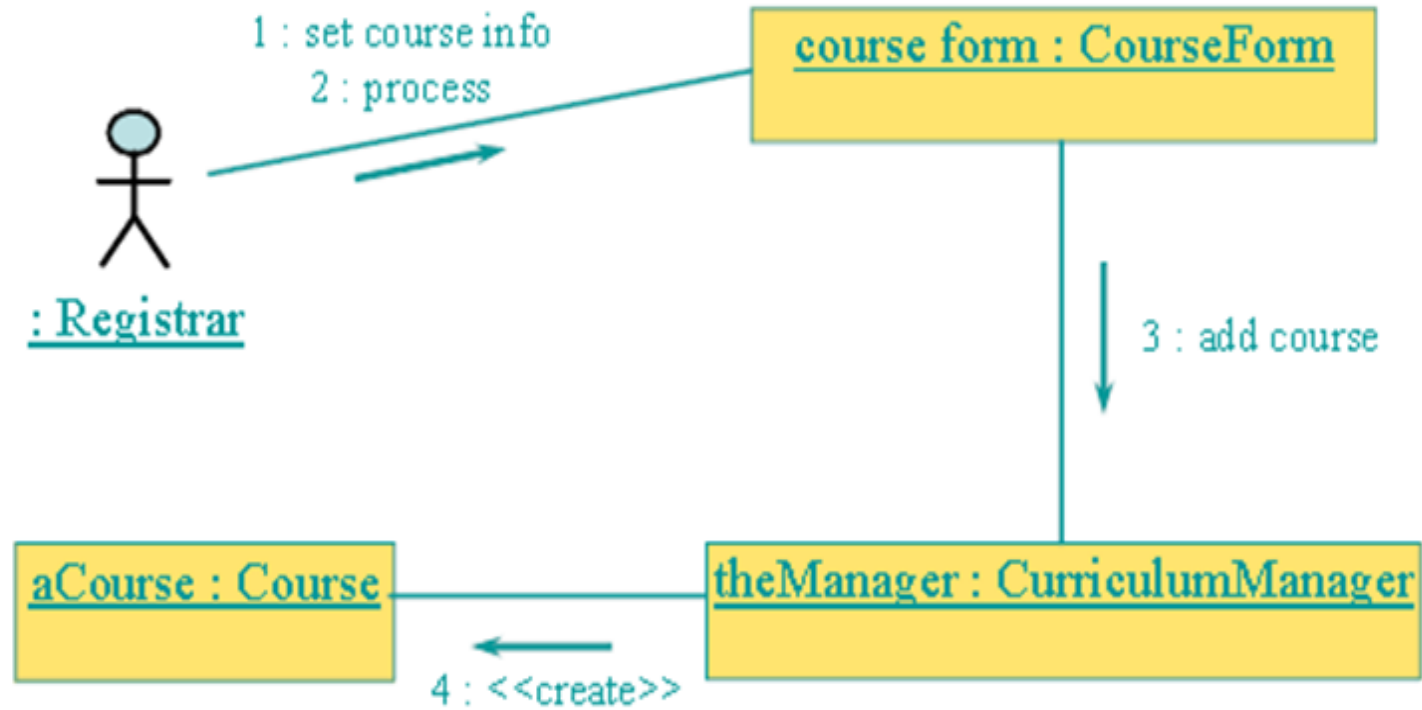


# CLASS ACTIVITY

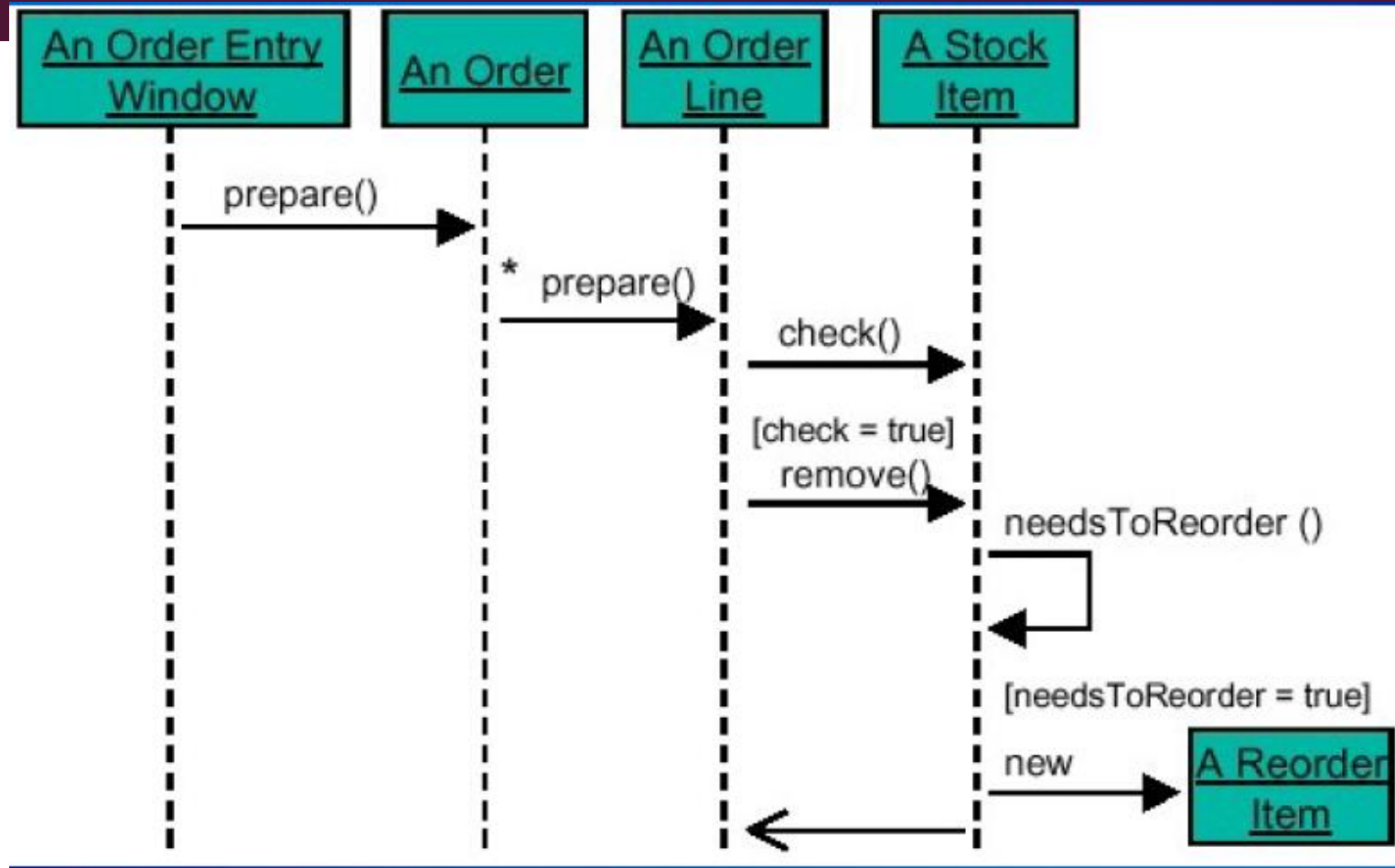
- Consider the software that controls a very simple cellular telephone. Such a phone has buttons for dialing digits, and a “send” button for initiating a call. It has “dialer” hardware and software that gathers the digits to be dialed and emits the appropriate tones. It has a cellular radio that deals with the connection to the cellular network. It has a speaker, and a display.
  
- **Use case: Make Phone Call**
  1. User presses the digit buttons to enter the phone number.
  2. For each digit, the display is updated to add the digit to the phone number.
  3. For each digit, the dialer generates the corresponding tone and emits it from the speaker.
  4. User presses “Send” for connecting to the network. The accumulated digits are sent to the network. The cellular radio establishes a connection to the network.
  5. The “in use” indicator is illuminated on the display
  6. The connection is made to the called party.



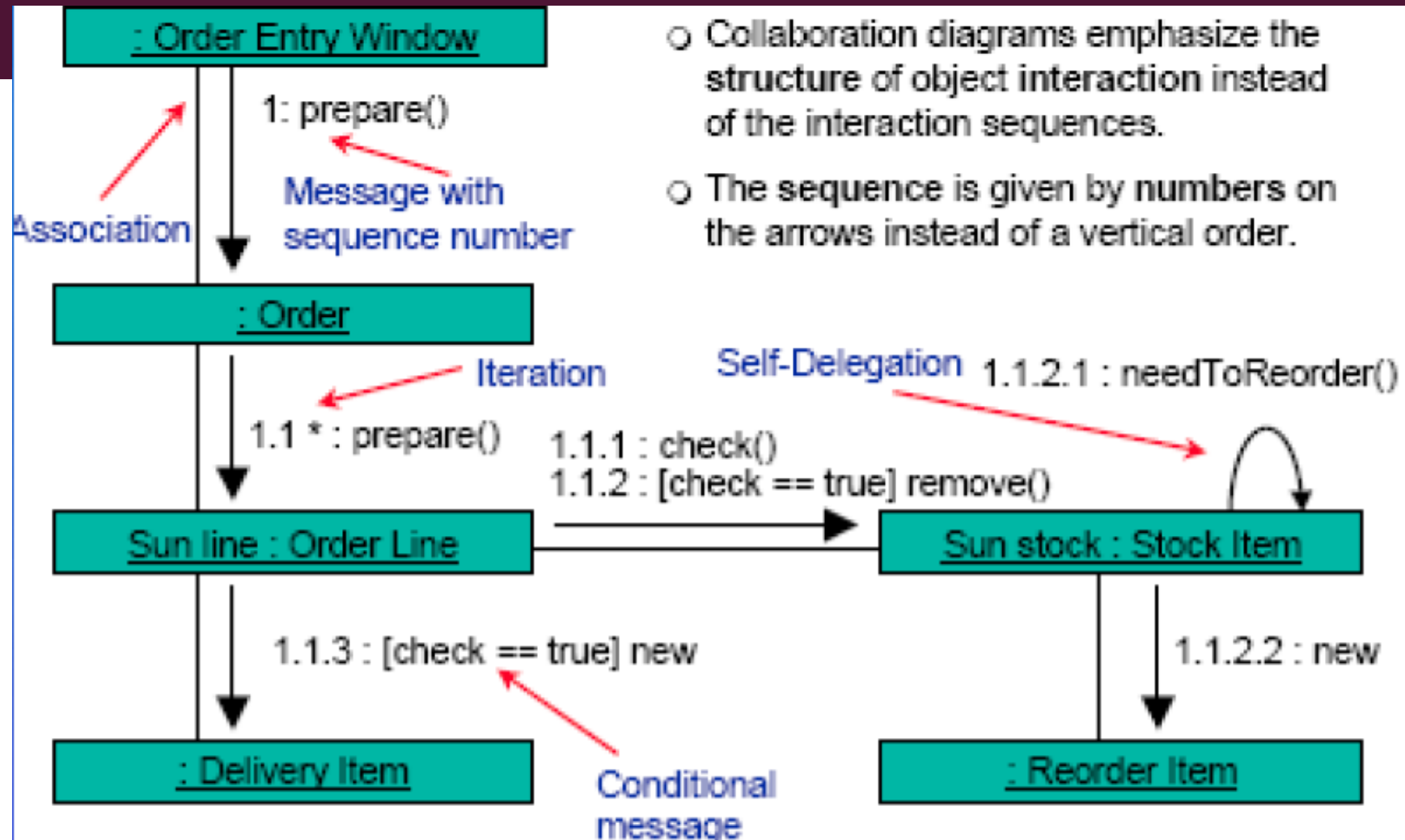
# Example



# EXAMPLE

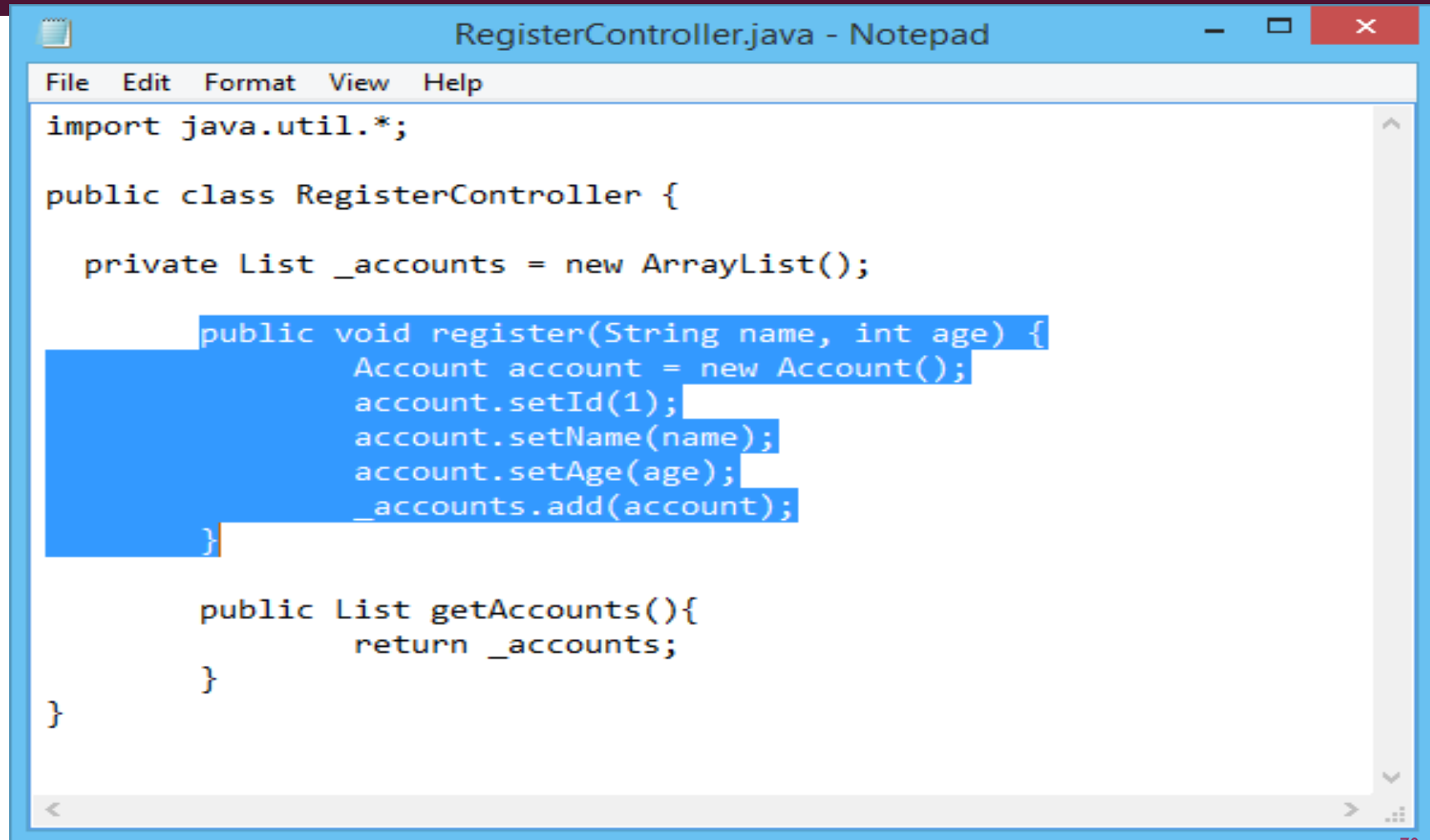


# EXAMPLE



- Collaboration diagrams emphasize the structure of object interaction instead of the interaction sequences.
- The sequence is given by numbers on the arrows instead of a vertical order.

# YOUR TURN



```
RegisterController.java - Notepad
File Edit Format View Help
import java.util.*;

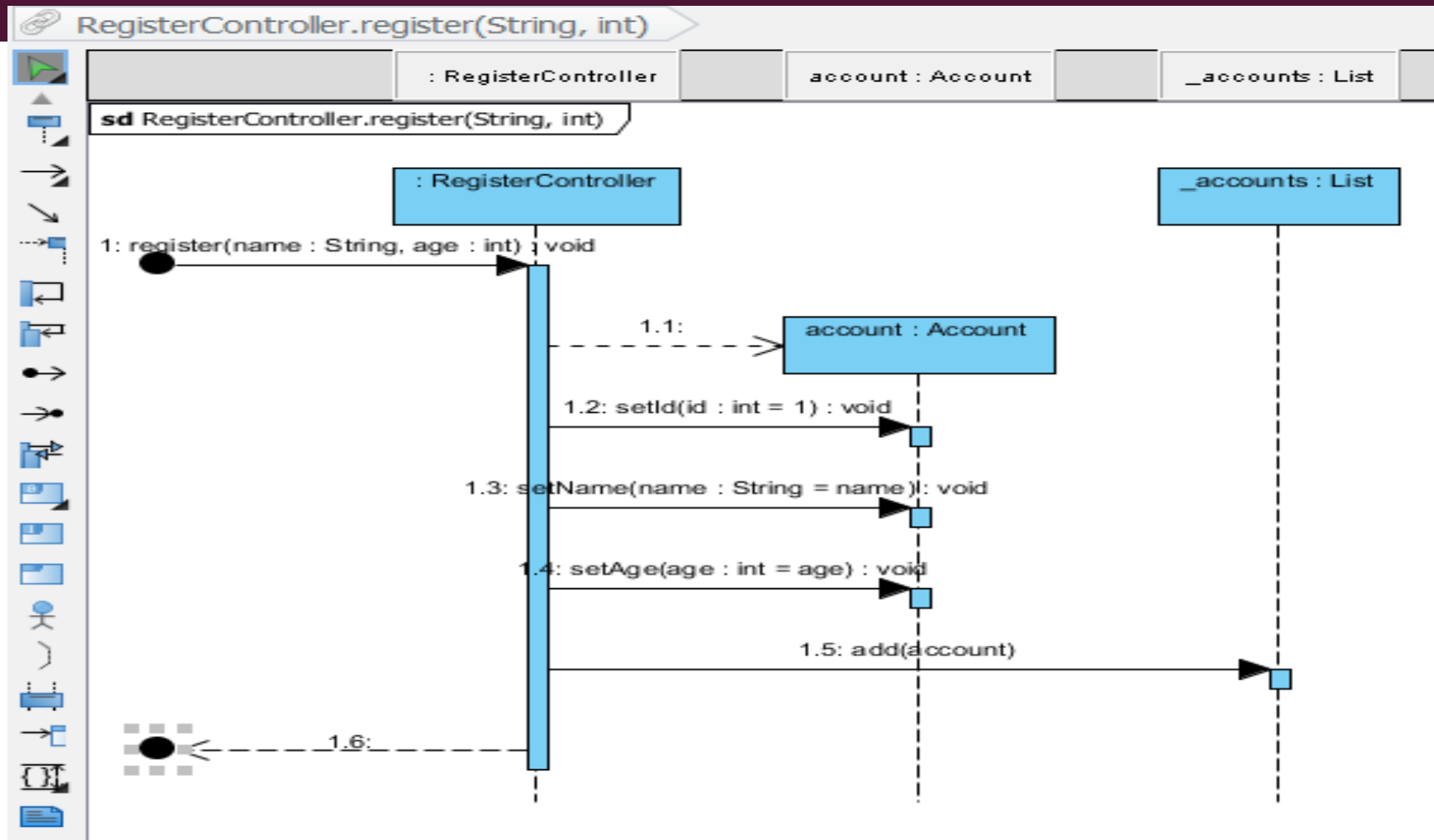
public class RegisterController {

    private List _accounts = new ArrayList();

    public void register(String name, int age) {
        Account account = new Account();
        account.setId(1);
        account.setName(name);
        account.setAge(age);
        _accounts.add(account);
    }

    public List getAccounts(){
        return _accounts;
    }
}
```

# SOLUTION





That is all