Object Oriented Analysis & Design

Engr. Abdul-Rahman Mahmood
DPM, MCP, QMR(ISO9001:2000)

armahmood786@yahoo.com
alphapeeler.sf.net/pubkeys/pkey.htm
pk.linkedin.com/in/armahmood
www.twitter.com/alphapeeler
www.facebook.com/alphapeeler
abdulmahmood-sss   alphasecure
armahmood786@hotmail.com
http://alphapeeler.sf.net/me

alphasecure@gmail.com
http://alphapeeler.sourceforge.net
http://alphapeeler.tumblr.com
armahmood786@jabber.org
alphapeeler@aim.com
mahmood_cubix   48660186
alphapeeler@icloud.com
http://alphapeeler.sf.net/acms/

interaction diagrams

Sequence and collaboration diagrams

What are interaction diagrams?

- Interaction diagrams illustrate how objects interact via messages in order to fulfil certain tasks.
- There are two kinds of interaction diagrams:

  - collaboration diagrams

  - sequence diagrams

Collaboration and sequence - differences

- 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.

Interaction diagrams: when and how to use them?

1. Identify the system events that are implied by the use cases.

2. Make at least one interaction diagram for each system event.

3. Make additional interaction diagrams for alternative courses of events.

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

michael:Person

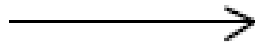michael

Types of message flows

- Synchronous: the sender waits until the responder finishes.

- Flat: 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.

Collaboration diagrams

- A collaboration diagram shows the <u>relationship between objects</u> and the <u>order of messages</u> passed between them.

- The objects are listed as icons and arrows indicate the messages being passed between them.

- The numbers next to the messages are called <u>sequence numbers</u> and, as the name suggests, they show the sequence of the messages as they are passed between the objects.
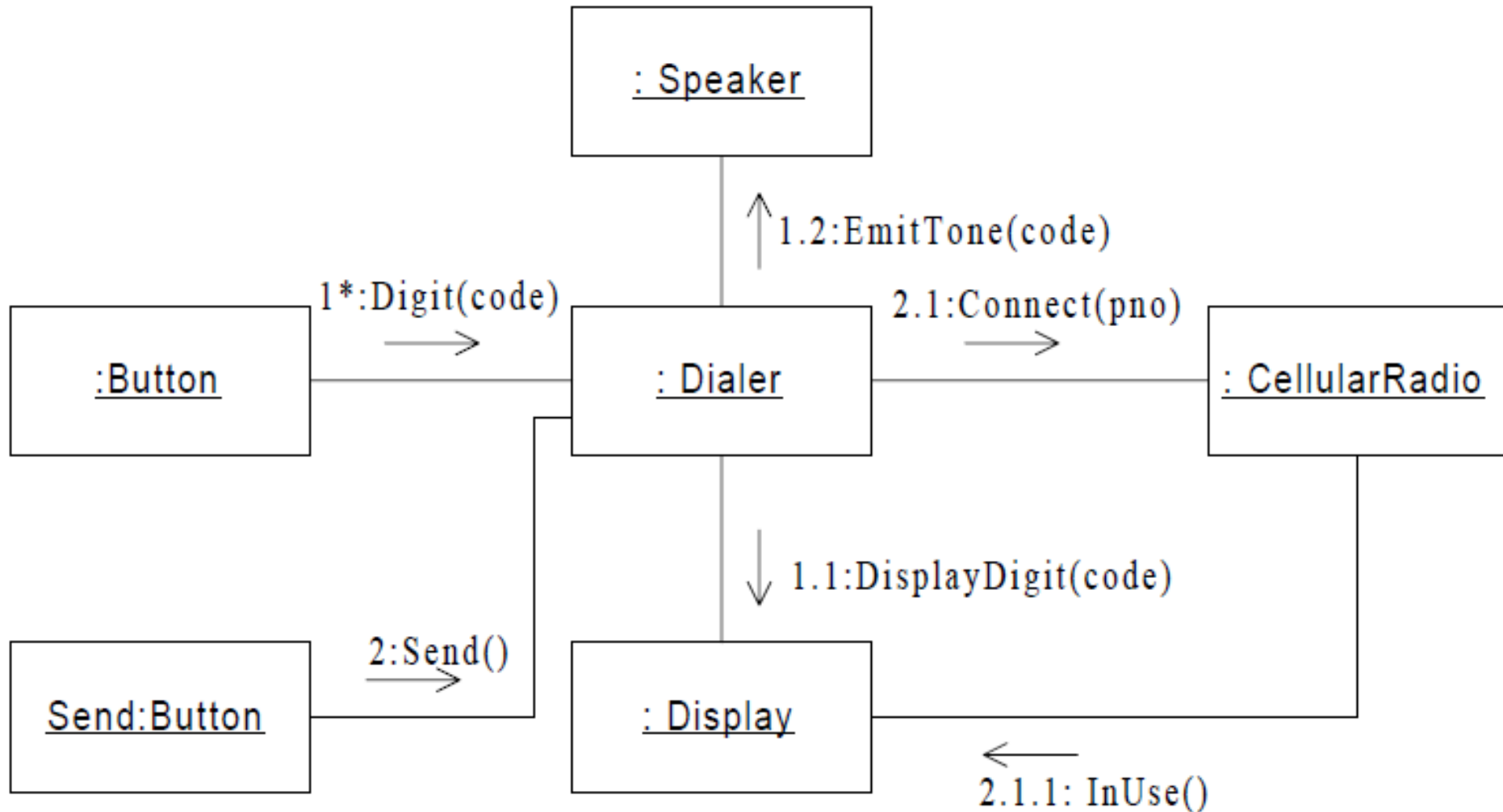
Collaboration diagram – an example
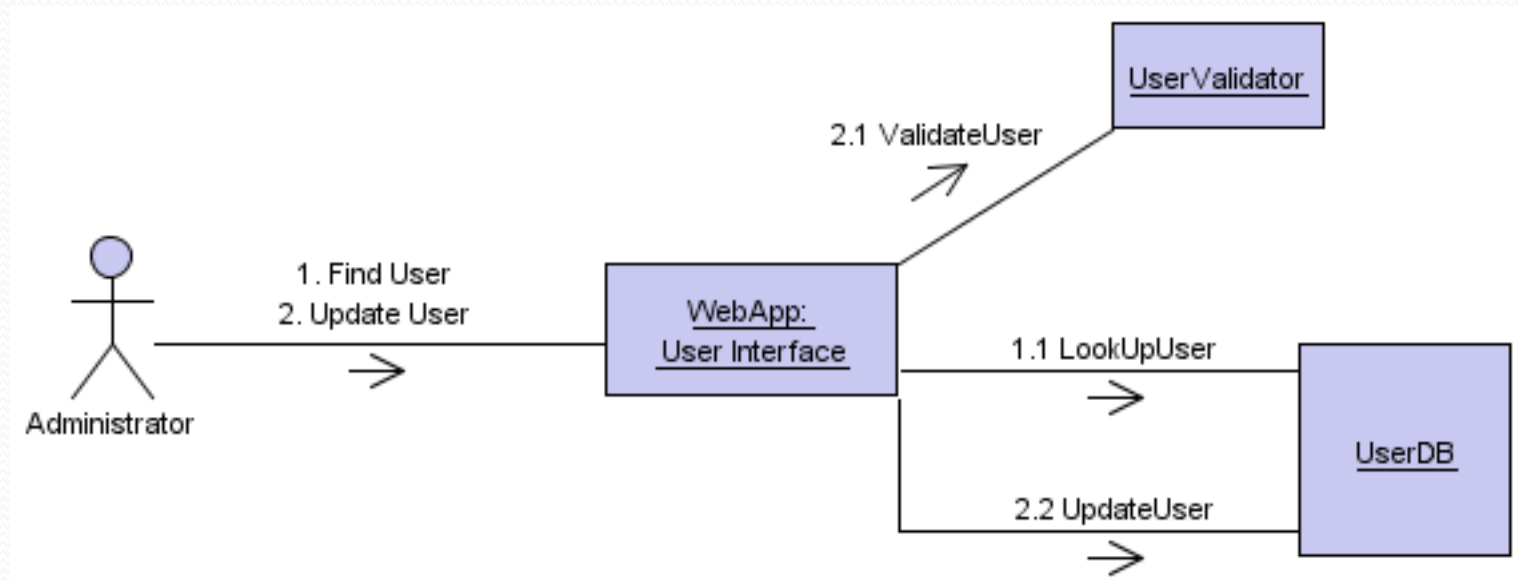
Example: A Cellular Phone

- 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 microphone, 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" 5. The "in use" indicator is illuminated on the display 6. The cellular radio establishes a connection to the network. 7. The accumulated digits are sent to the network. 8. The connection is made to the called party.

Collaboration diagrams of **Make Phone Call** use case.

Message 1*:Digit(code) has an asterisk in order to denote that it may occur many times before message 2.

Static model based on previous collaboration diagram
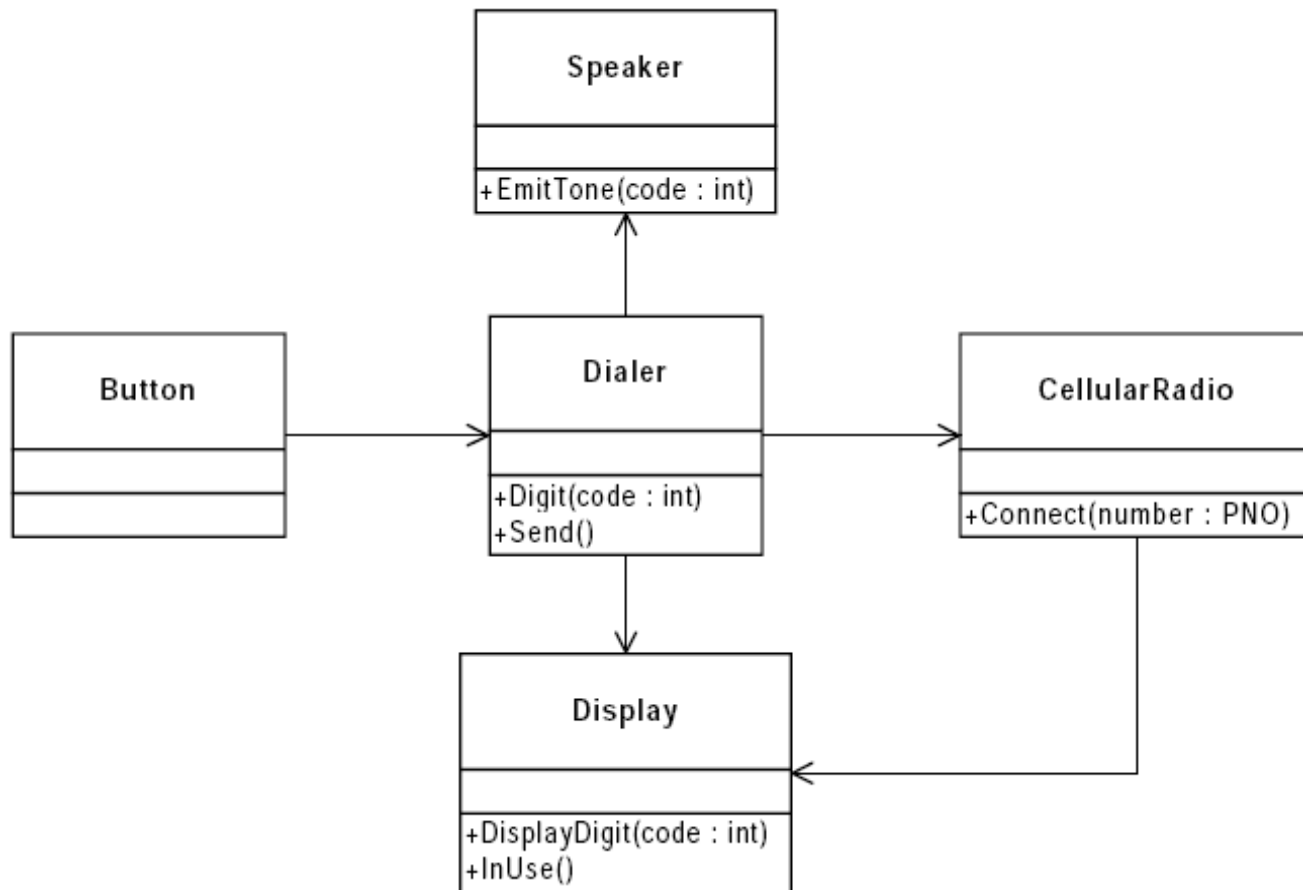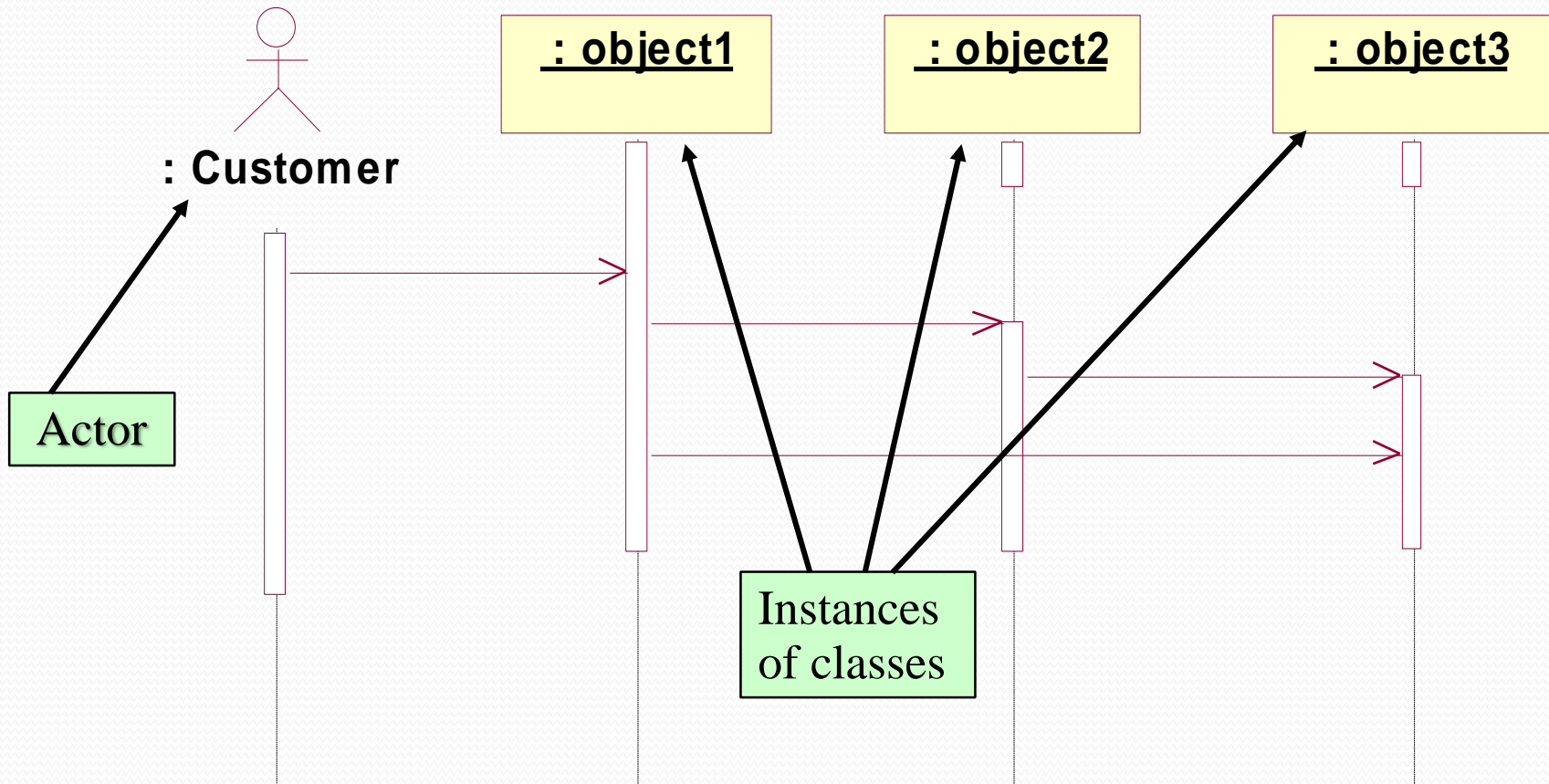
Collaboration diagrams – strenghts and weaknesses

One of the greatest strengths of collaboration diagrams is their simplicity.

The principal weakness, however, is that although they are good at describing behavior, they do not define it. They typically do not show all the iteration and control that is needed to give an computationally complete description.

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.

Sequence diagrams: an example

: object1   : object2   : object3

: Customer

Actor

Instances
of classes

Sequence Diagram: Lifelines

:actor

:object1

:object2

:object3

Time

Each object has a lifeline.

Time flows from top to down.
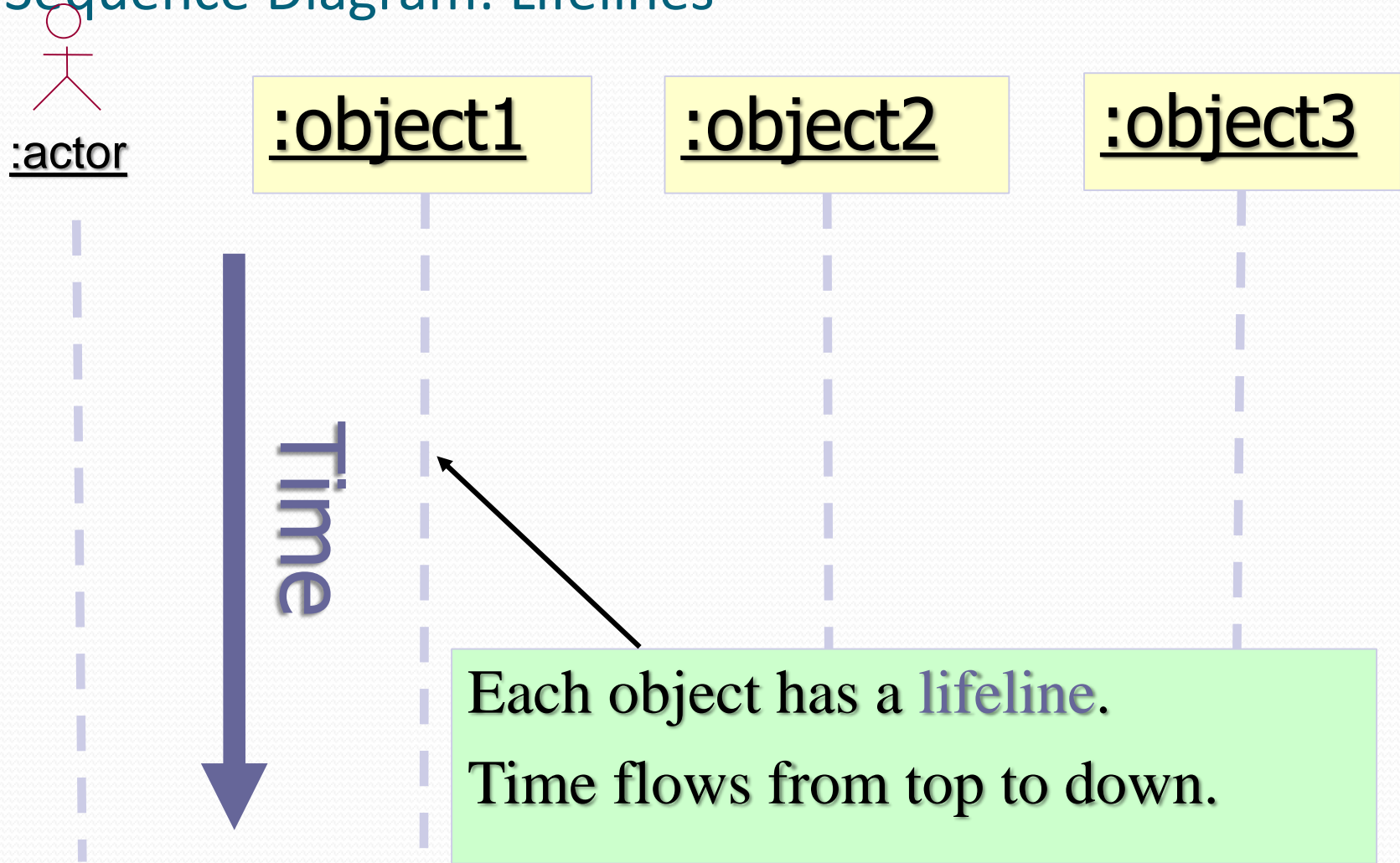
Sequence diagrams: rectangles

:actor

:object1

:object2

:object3

Time

Rectangles refer to events that are related to each other.
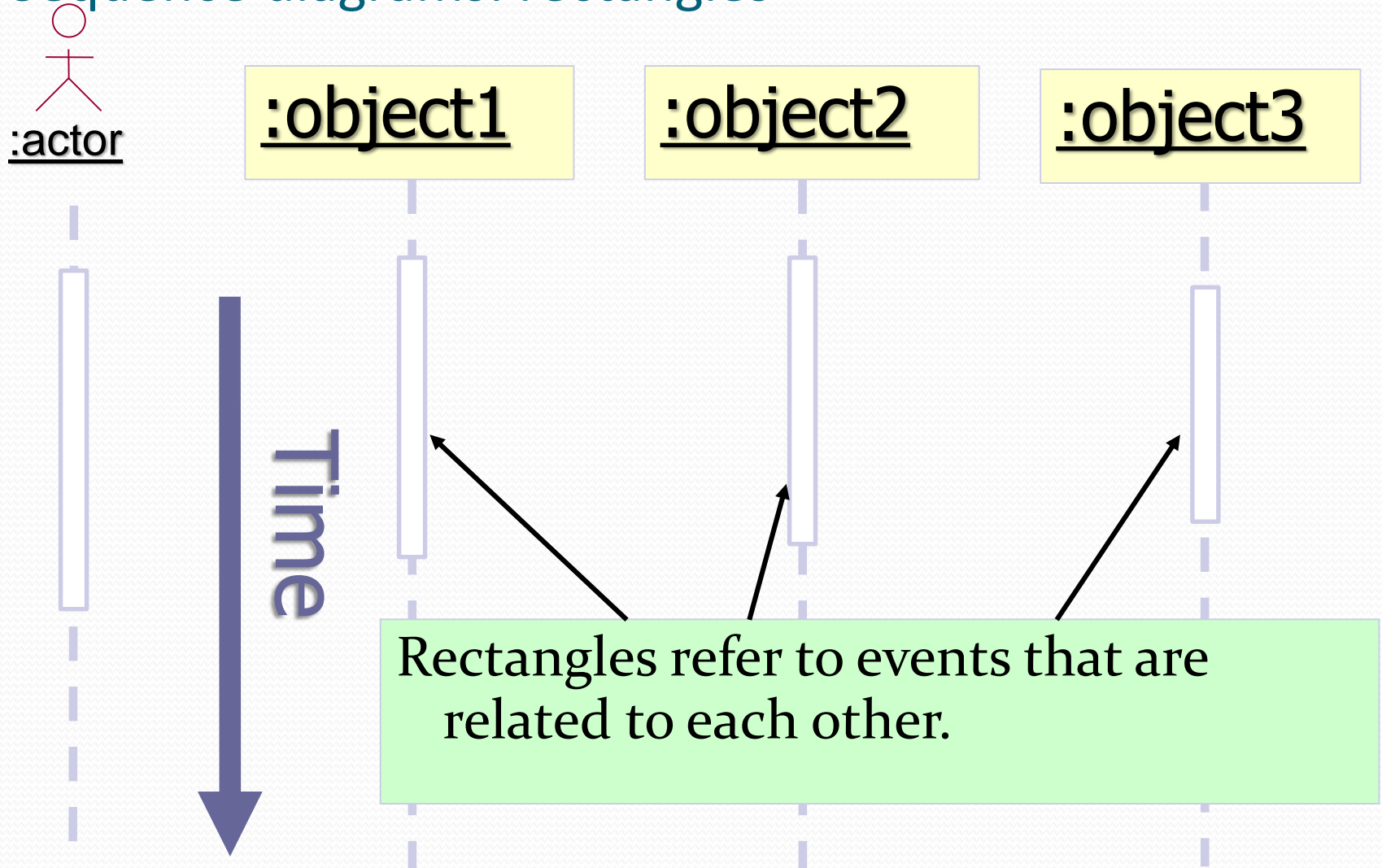
Sequence diagrams: messages

:actor

:object1

:object2

:object3

Time

Arrows indicate messages that are sent from one object to another.
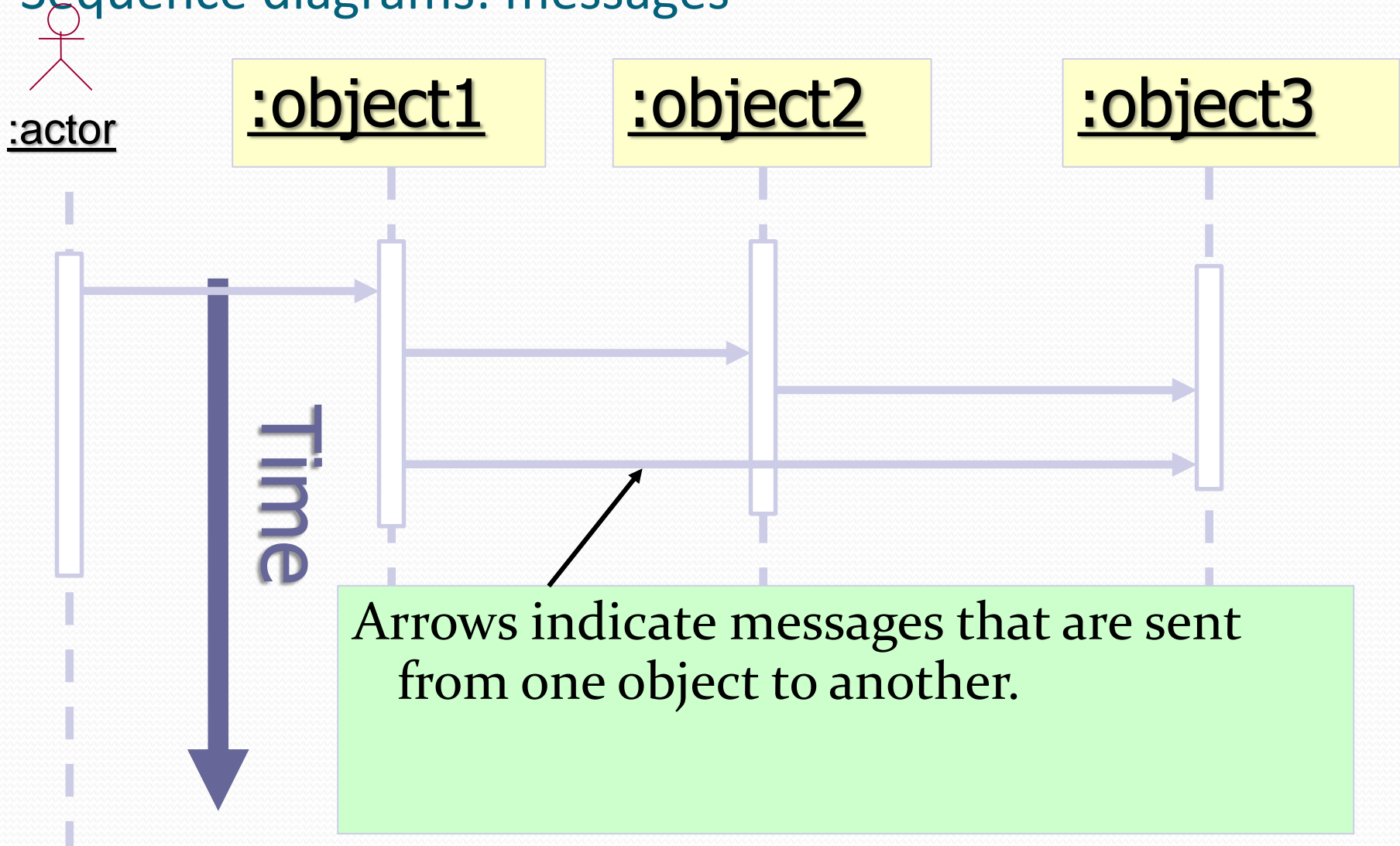
Sequence diagrams - different types of messages

: object1

: object2

: object3

: Customer

request( )

Message

createObject3( )

create( )

An object sends a message to itself.

Feedback arrow shows that a value is returned.
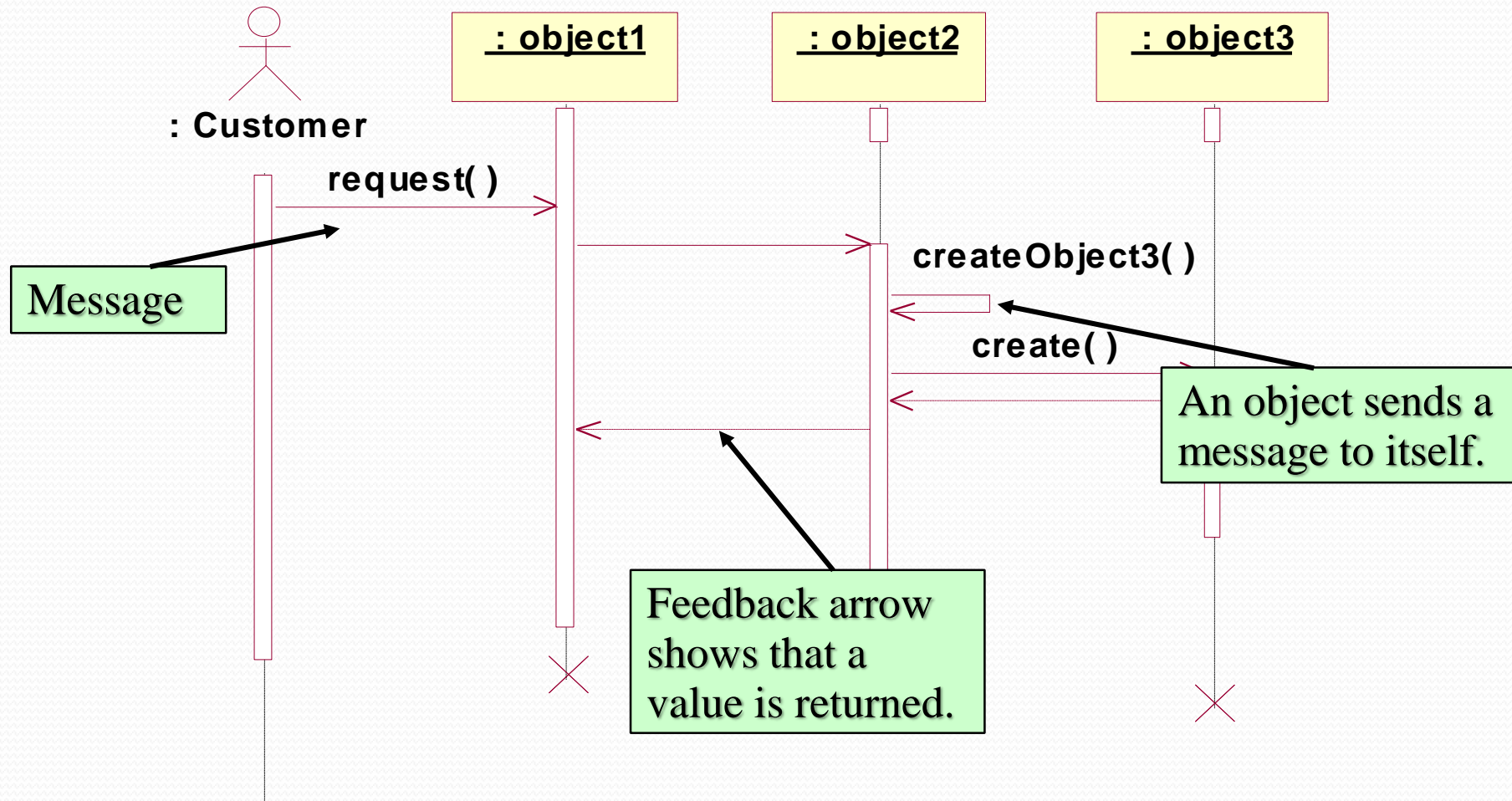
Sequence diagrams: endpoints

: Customer

: object1

: object2

: object3

request( )

createObject3( )

create( )

X at the end of the lifeline shows that the object ceases to exist.
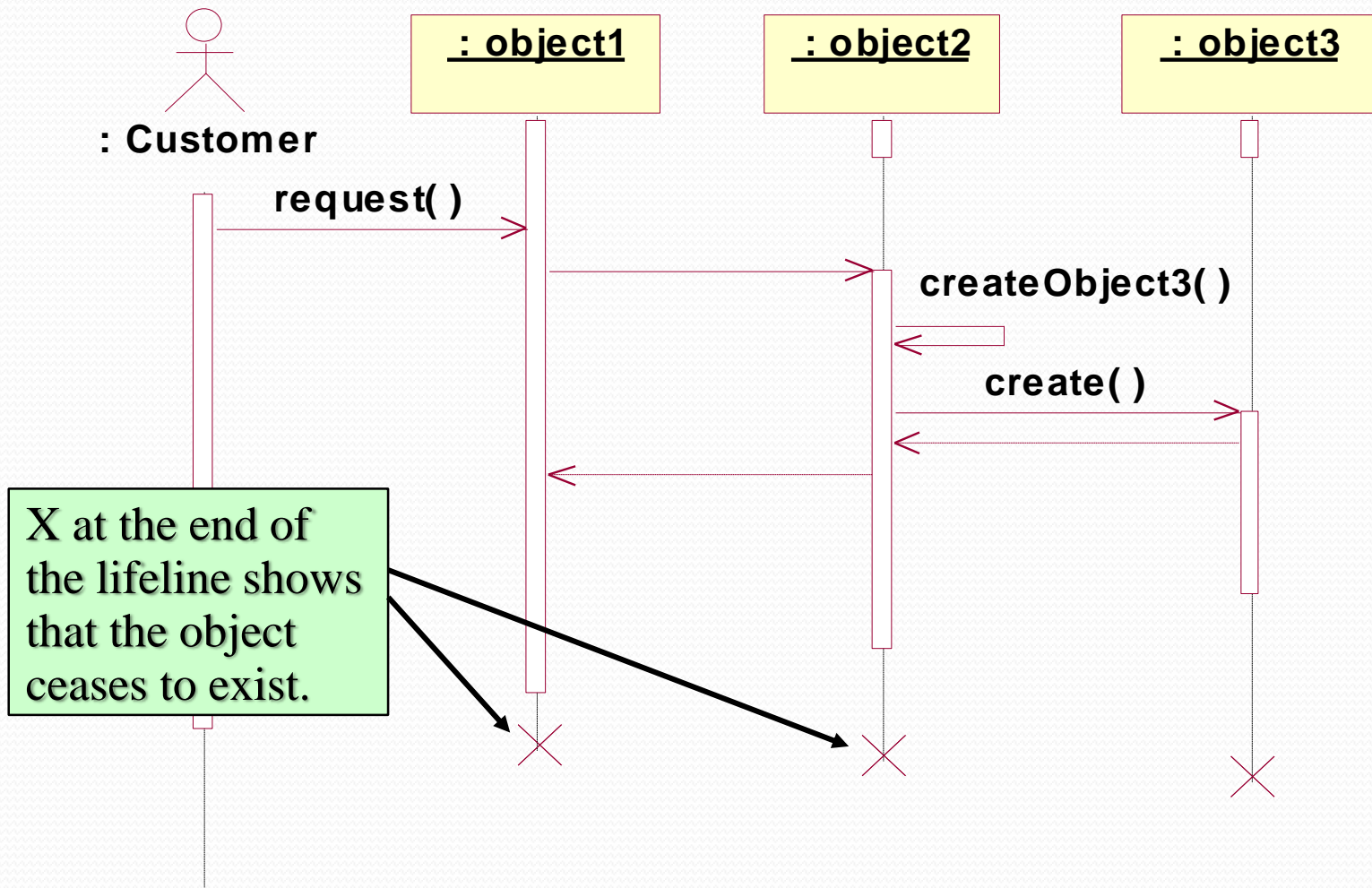
Sequence diagram: a more complex example
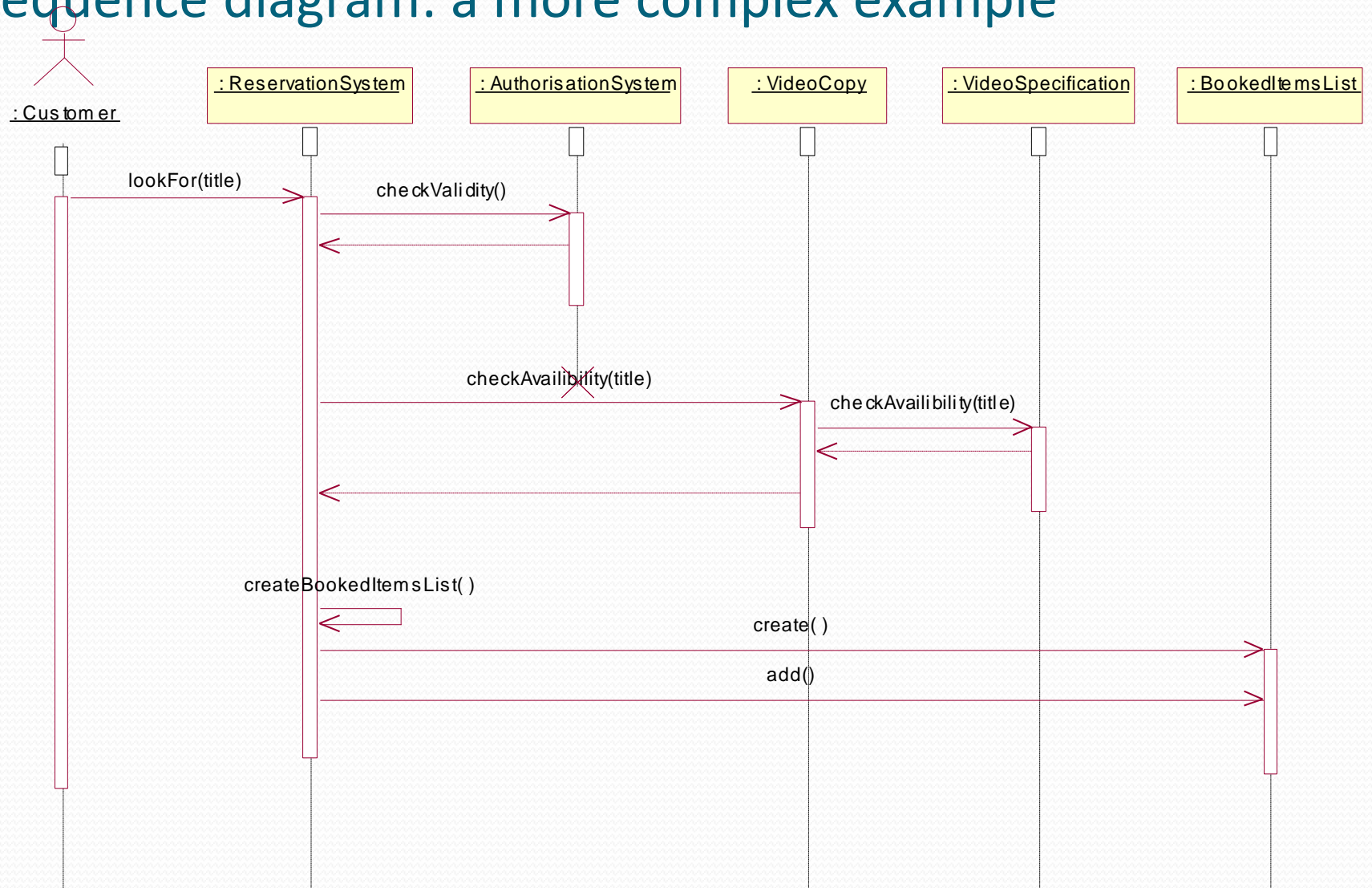
Sequence diagrams - strengths and weaknesses

Strengths:

- they clearly show sequence or time ordering of messages

- the notation is rather simple

Weaknesses:

- forced to extend to right when adding new objects, consumes a lot of space

UML Collaboration Diagram

Collaboration Diagrams

- Show a particular sequence of messages exchanged between a number of objects
- Use sequence diagrams to model flows of control by time ordering
  - Better for demonstrating the ordering of the messages
  - Not suitable for complex iteration and branching
- Use collaboration diagrams to model flows of control by organization
  - Good at showing the static connections among the objects while demonstrating a particular sequence of messages at the same time

Collaboration Diagrams

- Shows the relationship between objects and the order of messages passed between them between them.

- The objects are listed as rectangles and arrows indicate the messages being passed.

- The numbers next to the messages are called <u>sequence numbers</u>. They show the sequence of the messages as they are passed between the objects convey the same information as sequence diagrams, but focus on object roles instead of the time sequence.
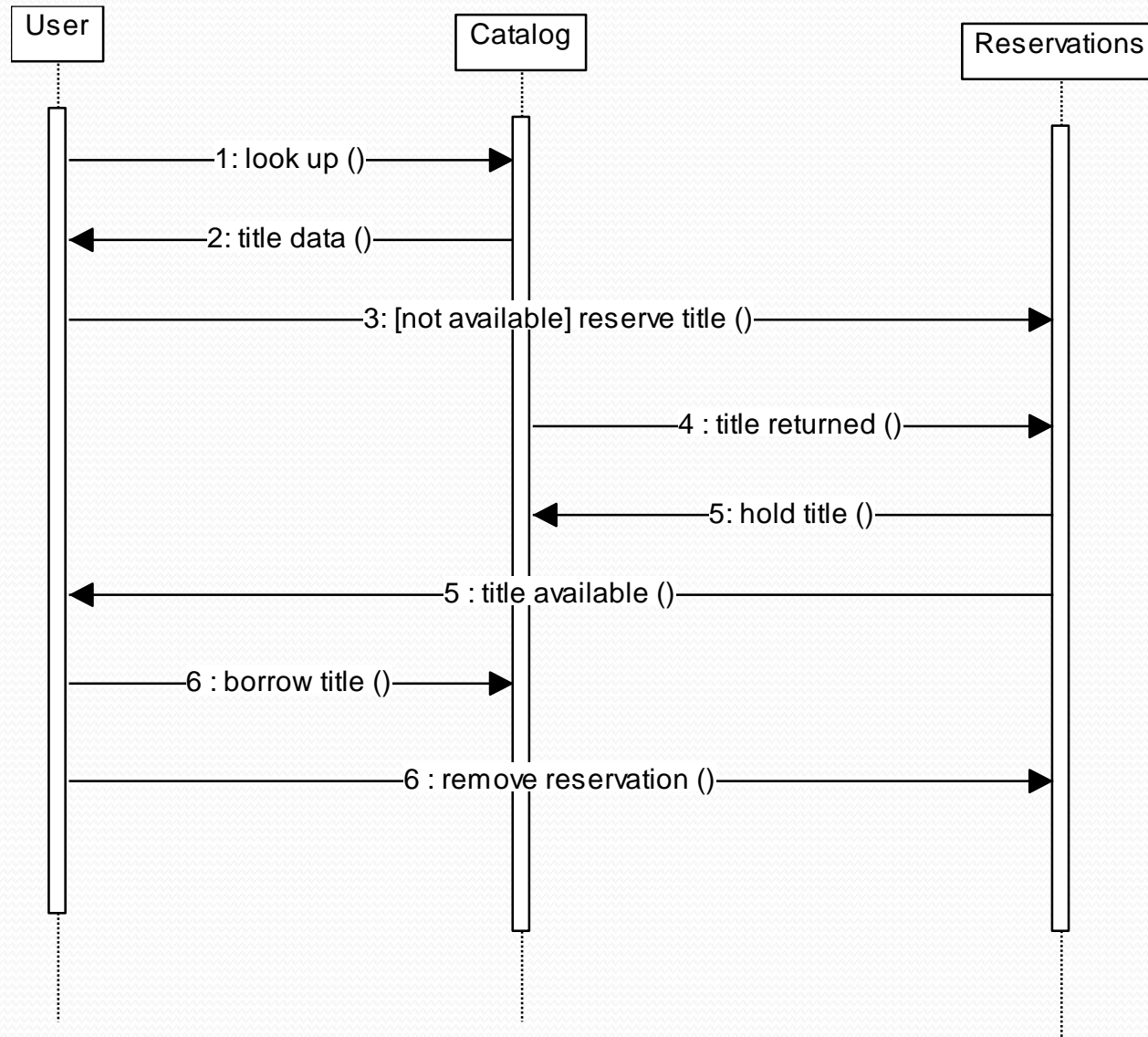
Collaboration Diagrams

Collaboration Diagrams describe interactions among classes and associations. These interactions are modeled as exchanges of messages between classes through their associations. Collaboration diagrams are a <u>type of interaction diagram</u>. Collaboration diagrams contain the following <u>elements</u>.

• **Class roles**, which represent roles that objects may play within the interaction.

• **Association roles**, which represent roles that links may play within the interaction.

• **Message flows**, which represent messages sent between objects via links. Links transport or implement the delivery of the message.

Collaboration Diagrams

UML Collaboration diagrams (interaction diagrams) illustrate the relationship and interaction between software objects. <u>They require use cases, system operation contracts, and domain model to already exist.</u> The collaboration diagram illustrates messages being sent between classes and objects (instances). A diagram is created for each system operation that relates to the current development cycle (iteration).

Sequence Diagram (Issuing a book from library)

Example Sequence Diagram

:OrderEntryWindow	:Order	:ProductOrder	:StockItem

prepare()

prepare()

check()

[check="true"]
remove()

needsToReorder()

[needsToReorder="true"]
<<create>>

:ReorderItem

[check="true"]
<<create>>

:DeliveryItem

Collaboration Diagram

:OrderEntryWindow

object ←

:Order

link →

1:prepare() ← message

Sequence numbers are used to show the time ordering among the messages

sequence number

1.1.2.1:needsToReorder()

1.1.5:reply

1.1:prepare()

1.1.3: reply

:ProductOrder

:StockItem

1.1.1:check()
1.1.2:[check==true]remove()

1.1.4:new

1.1.2.2:[Reorder=true] create

:DeliveryItem

:ReorderItem

Collaboration Diagrams

- Class diagrams indicates what classes are part of our system, what they offer, how they relate, but they don't tell us how they communicate.

- Collaboration diagrams show (used to model) how objects interact and their roles.

- They are very similar to sequence diagrams. Actually they are considered as a **cross** between class and sequence diagram.

- *Sequence Diagrams* are arranged according to Time.

- *Collaboration Diagrams* represent the structural organization of object.

- [Both sequence and collaboration diagrams are called interaction diagrams]
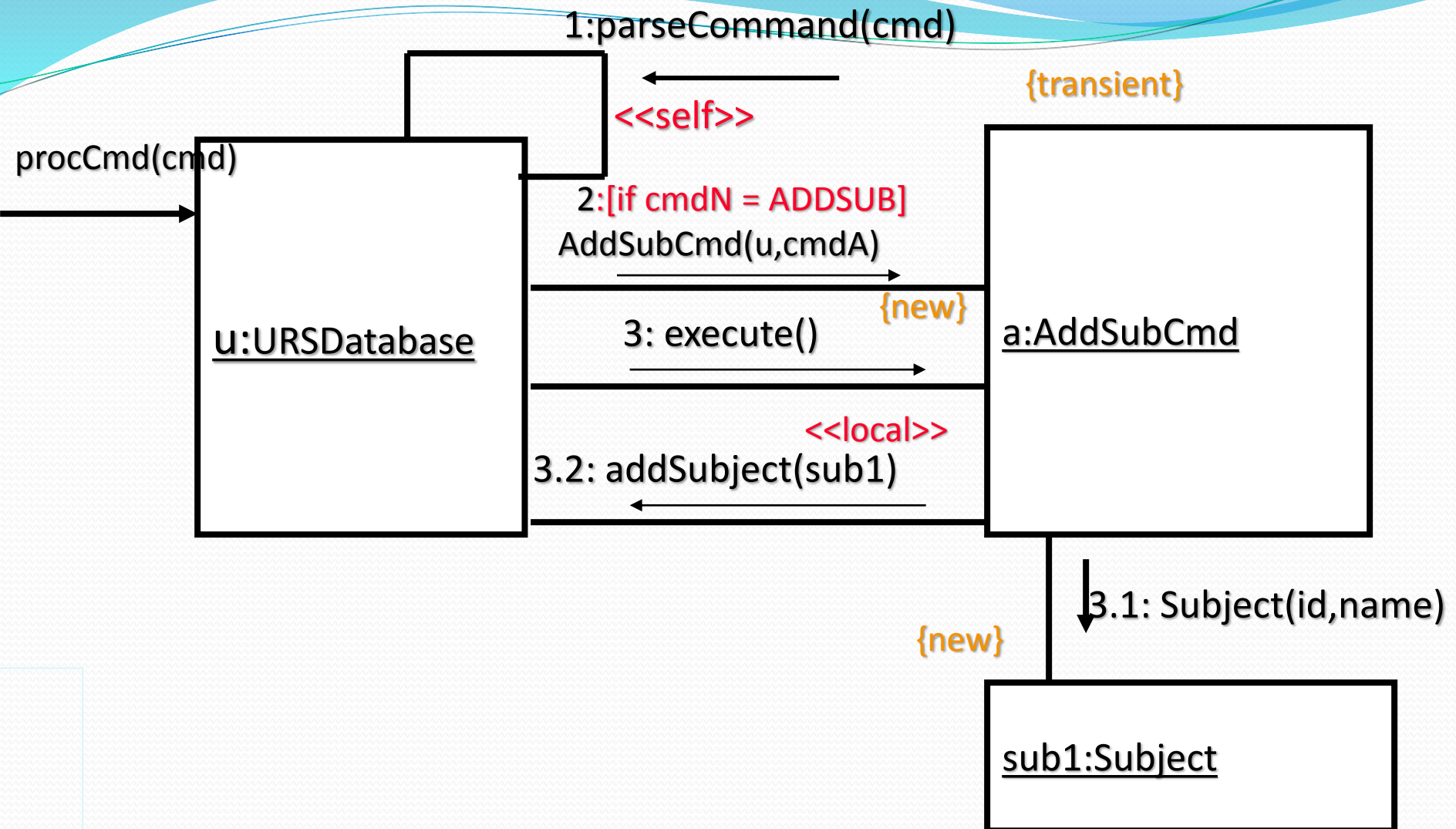
Collaboration Diagram – URS Add Subject Scenario

1:parseCommand(cmd)

{transient}

<<self>>

procCmd(cmd)

u:URSDatabase

2:[if cmdN = ADDSUB]
AddSubCmd(u,cmdA)

{new}

3: execute()

a:AddSubCmd

<<local>>
3.2: addSubject(sub1)

3.1: Subject(id,name)

{new}

sub1:Subject

Collaboration Diagram – URS Add Subject Scenario

<<self>>

1:parseCommand(cmd)

procCommand(cmd)
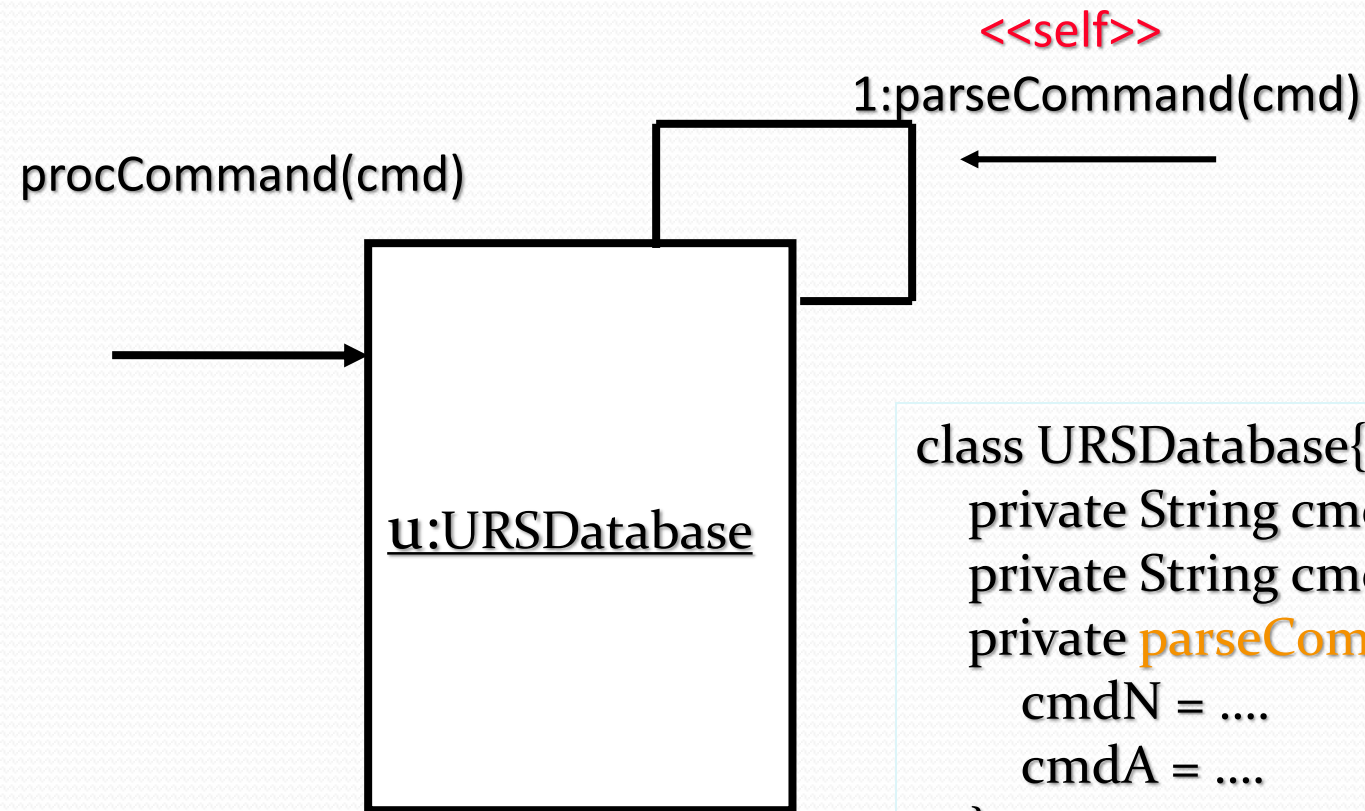
u:URSDatabase

```
class URSDatabase{
    private String cmdN;
    private String cmdA;
    private parseCommand(String cmd){
        cmdN = ....
        cmdA = ....
    }
    public procCommand(String cmd){
      parseCommand(cmd);
    }
}
```

Collaboration Diagram – URS Add Subject Scenario

u:URSDatabase

2: AddSubCmd(u,cmdA)

{transient}

a:AddSubCmd

{new}
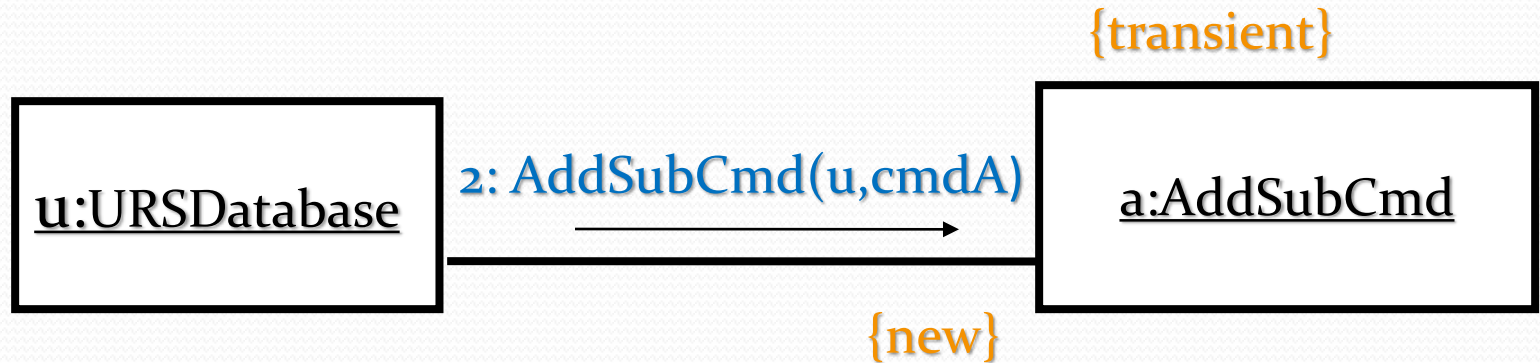
```
class URSDatabase{
  private String cmdN;
  private String cmdA;

  public procCommand(String cmd){
    parseCommand(cmd);
    if (cmdN == ADDSUB){
      AddSubCmd a = new AddSubCmd(u,cmdA);
    }
  }
}
```

Collaboration Diagram – URS Add Subject Scenario

Refer to class notes for further details .

Collaboration Diagrams

- Collaborations Diagrams show transient links that exists between objects.

  - <<self>> - A message from object to itself
  - << local>> - A message sent due to the object begin defined as a local variable in the method.
  - <<parameter>> - The object reference was sent as a parameter to the method.
  - <<global>> The object is global.

Collaboration Diagrams

- **<u>UML Collaboration Diagrams</u>**
  - Collaboration Diagram Semantics
  - Collaboration Diagram Notation
  - Collaboration Diagram Examples
  - Collaboration Diagram Issues

Collaboration Diagram Semantics

- Member of the Behavioral Group of diagrams as well as Sequence, State chart, and Activity diagrams

- **Behavioral description includes**

- **Static** - structural description on the participants

- **Dynamic** - description on the execution of the actions

- **Collaboration** – set of participants and interactions which are meaningful in a context

Collaboration Diagram Semantics

- **Purpose** – models the exchange of messages between objects to achieve something

- **Can be attached to**: operation or use case class

- **Parameterized collaboration** – can be reused can assist in defining the structural aspects of a design pattern
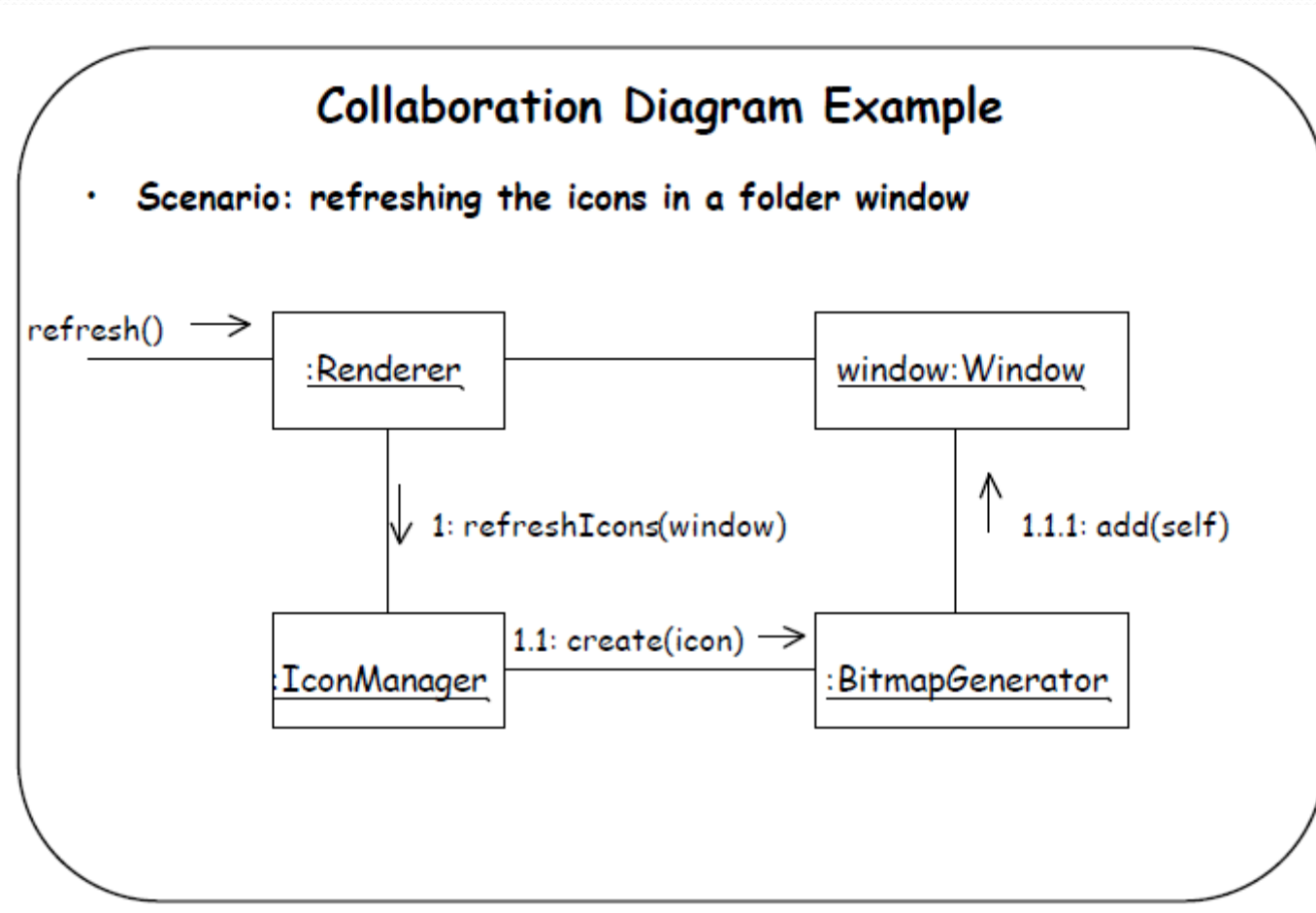
Collaboration Diagram Notation

- **<u>Represents a Collaboration and Interaction</u>**
- **Collaboration** - set of objects and their interactions in a specific context
- **Interaction** - set of messages exchanged in a collaboration to produce a desired result
- **Objects** – rectangles containing the object signature – object signature:

  object name : object Class • object name (optional) - starts with lowercase letter   class name (mandatory) - starts with uppercase letter – objects connected by lines – user (stick man) can appear

Messages

- **Messages** – are labeled like C and Java function calls followed by round brackets, and can have parameters and return values are followed by an arrow to show direction internal messages are numbered, starting from 1

Collaboration Diagram Example

Collaboration Diagram Example

- Scenario: refreshing the icons in a folder window

refresh() → :Renderer ——— window:Window

1: refreshIcons(window)

1.1.1: add(self)

:IconManager —— 1.1: create(icon) → :BitmapGenerator

Collaboration Diagram Issues
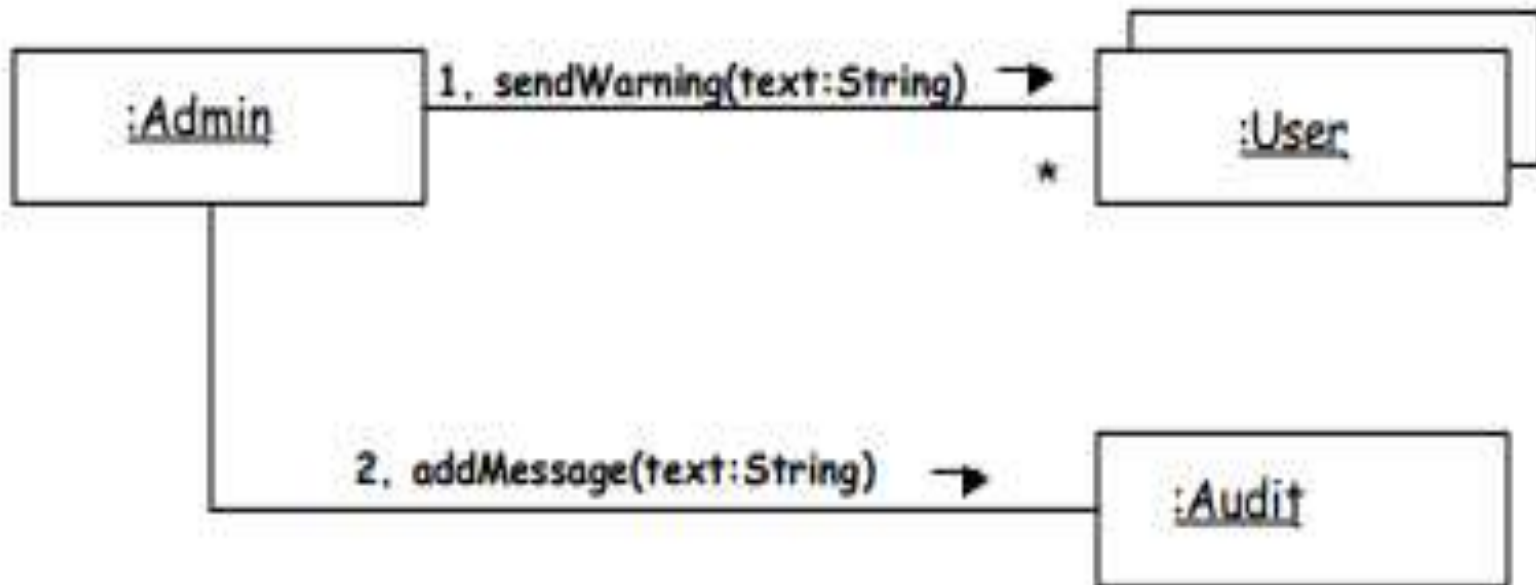
Message Signature

- **Guard**
  - condition applied to the message
  - in square brackets at the start of the signature
- **Sequence number**
  - numbers separated by dots, ending in a colon
- **Return value**
  - name followed by :=

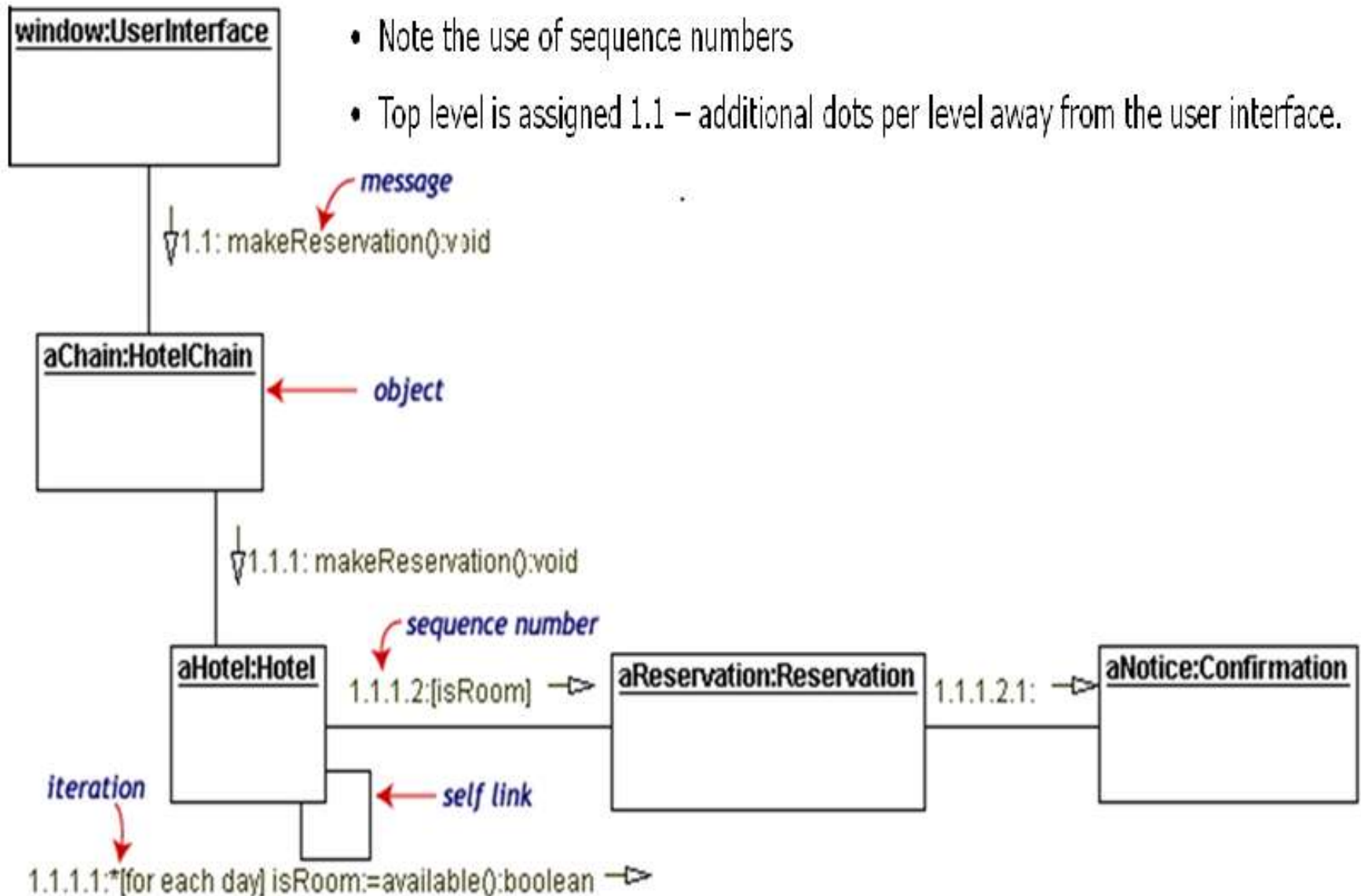# Collaboration Diagram Issues

- **Operation name**
- **Argument list**
  - names separated by commas, within round brackets
  - Types of message flows
- **Synchronous, asynchronous, simple**
  - Multiplicity of objects
- **How can messages sent to a multiplicity (e.g., an array) of objects be represented?**

Multiplicity in Collaboration Diagrams

- A server object needs to send a message to each user logged in, and another message to the audit object (which keeps a track of messages sent)

Collaboration – Hotel reservation system

window:UserInterface

- Note the use of sequence numbers
- Top level is assigned 1.1 – additional dots per level away from the user interface.

message

1.1: makeReservation():void

aChain:HotelChain

object

1.1.1: makeReservation():void

sequence number

aHotel:Hotel 1.1.1.2:[isRoom] ➞ **aReservation:Reservation** 1.1.1.2.1: ➞ **aNotice:Confirmation**

iteration

self link

1.1.1.1:*[for each day] isRoom:=available():boolean ➞