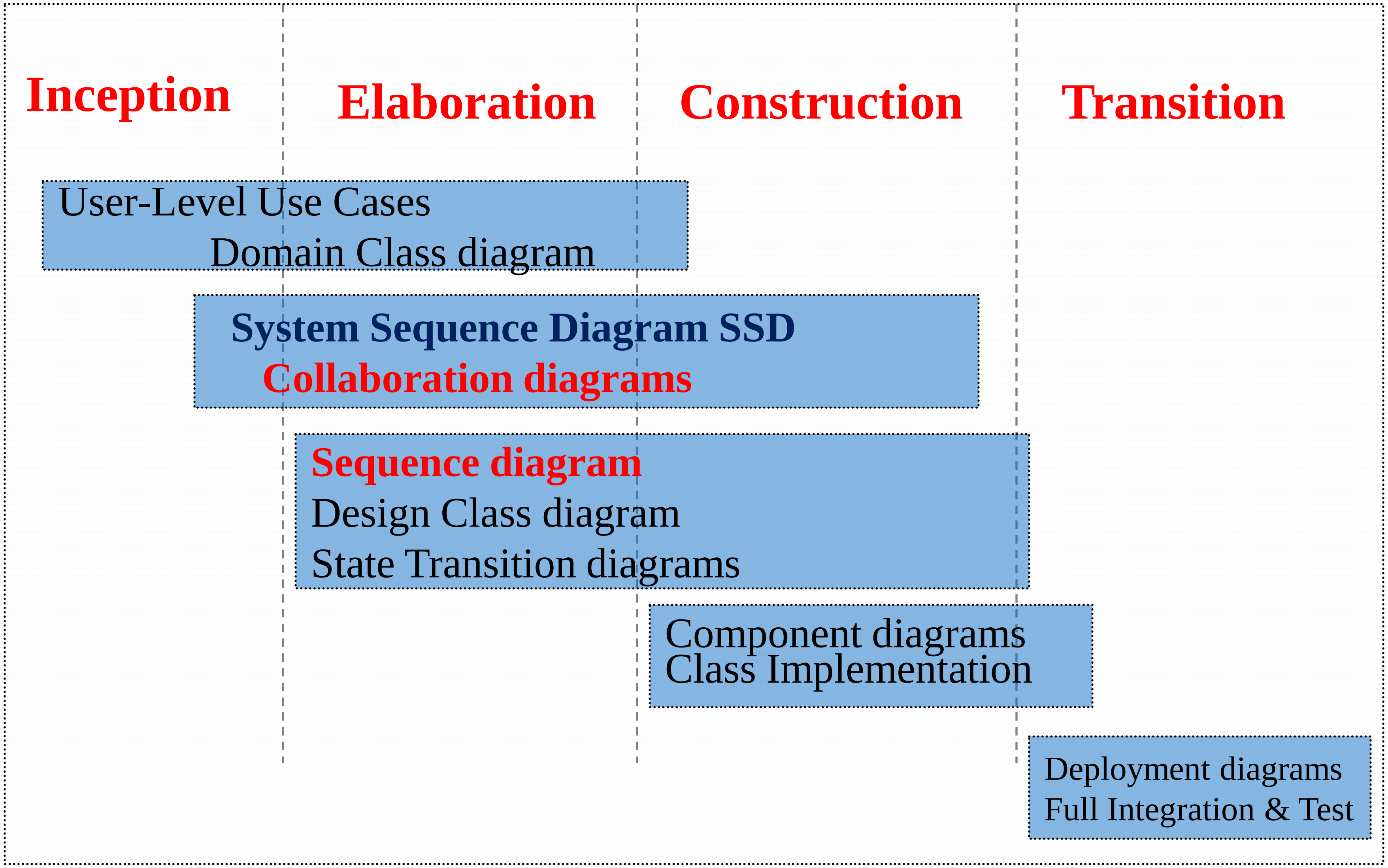
COMP 3711

(OOA and OOD)

Dynamic Object Modeling Interaction Diagrams

Larman Chapter 14, 15

UML And UP



Inception
Elaboration
Construction
Transition
User-Level Use Cases
Domain Class diagram
System Sequence Diagram SSD
Collaboration diagrams
Sequence diagram
Design Class diagram
State Transition diagrams
Component diagrams
Class Implementation
Deployment diagrams
Full Integration & Test

Design - Think Object

- *What are the responsibilities of the object?*
- *Who does it collaborate with?*
- *What design patterns should be applied?*
- *Draw then Code*

UML Dynamic Modeling

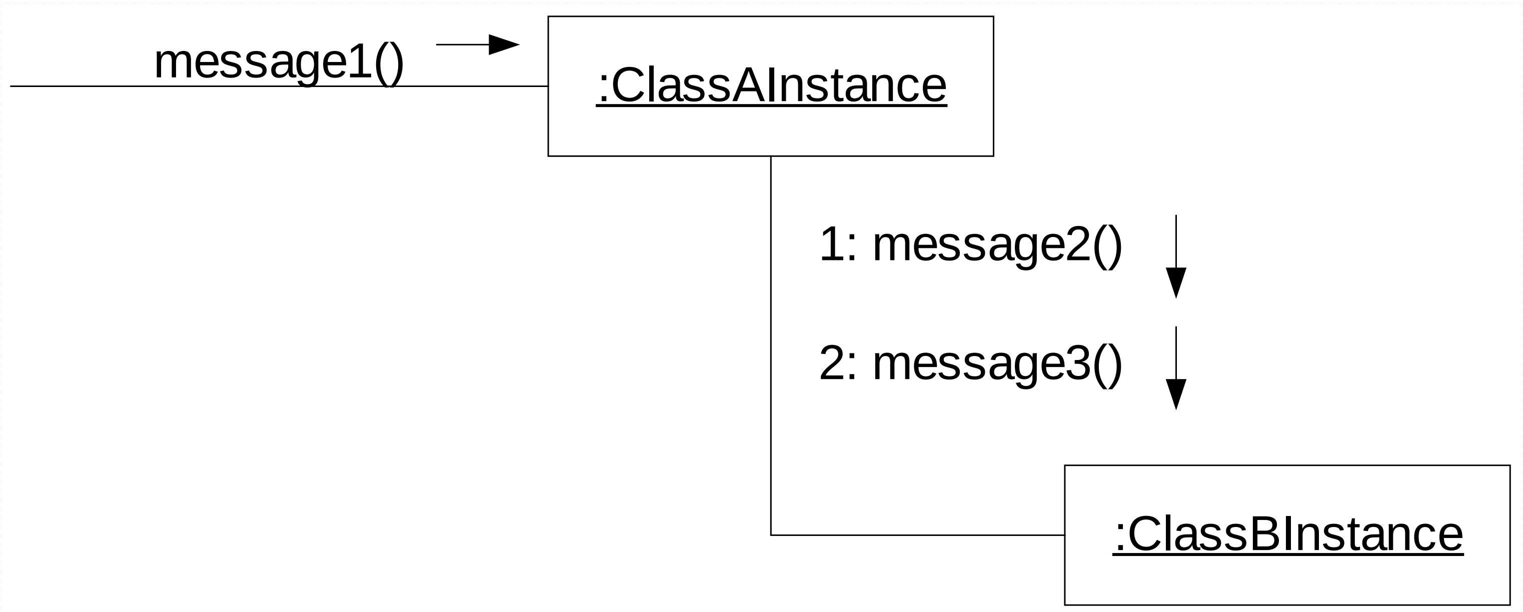
- Illustrate how objects interact via messages
- Interaction diagram is a generalization of two more specialized UML diagram types:

1. Collaboration (Communication) Diagram
2. Sequence Diagram

Both express similar message interactions

Collaboration (Communication) Diagram

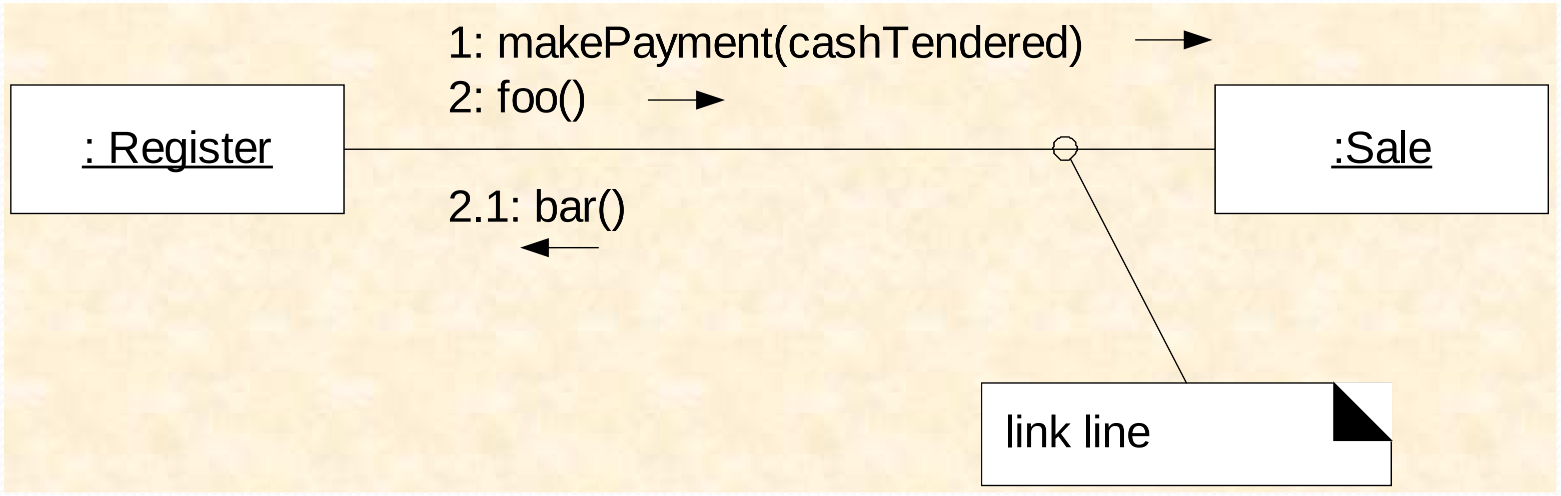
- Show object interactions in a network format
- Objects can be placed anywhere on the diagram
- Good for sketching model on walls – e.g. Agile Modeling practices



Collaboration Diagram Notations

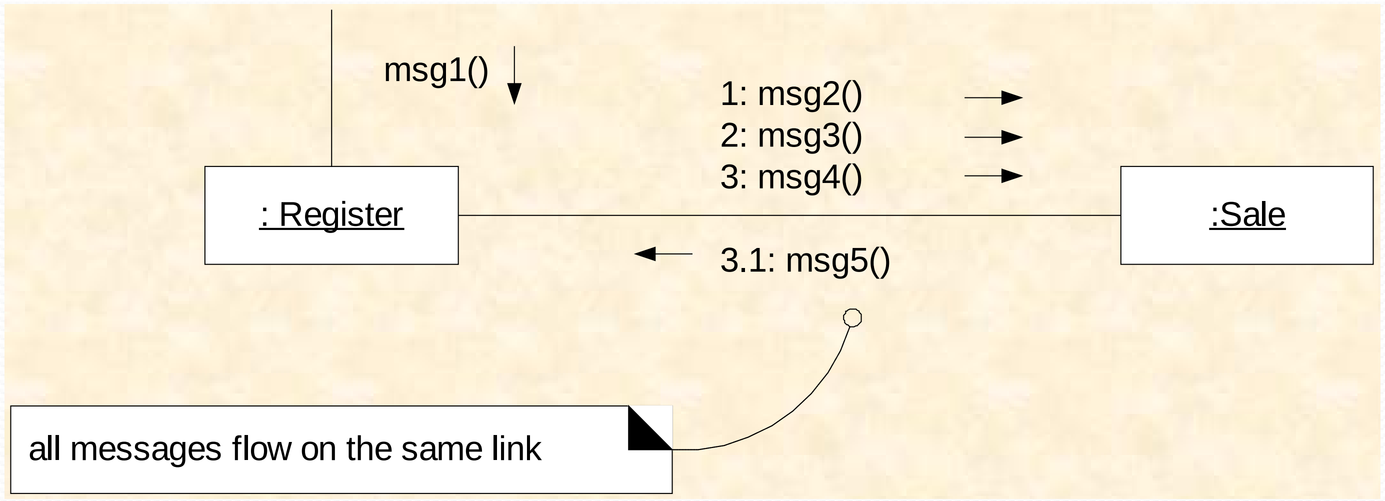
Link:

- A connection path between two objects
- Indicates some form of navigation and visibility between the objects is possible
- Multiple messages can flow along the same single link

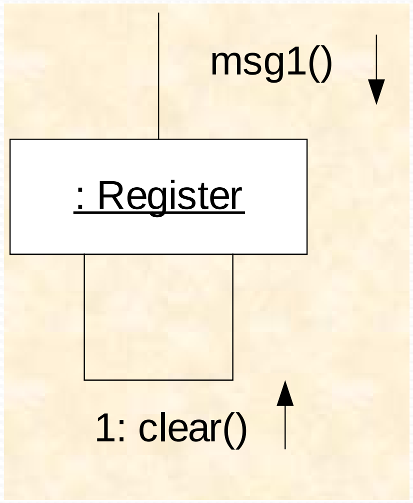


Collaboration Diagram Notations

- All messages flow on same link

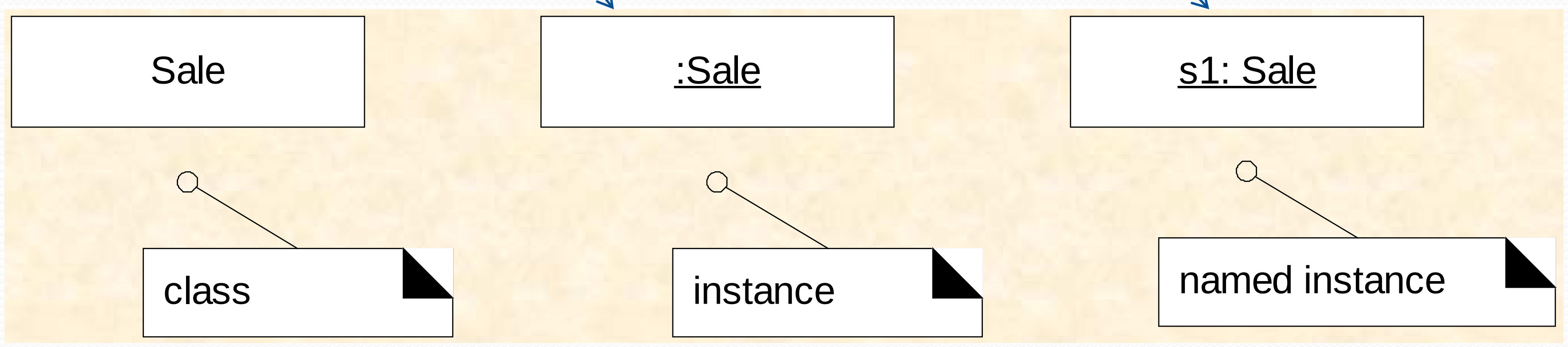


- Message can be sent from an object to itself



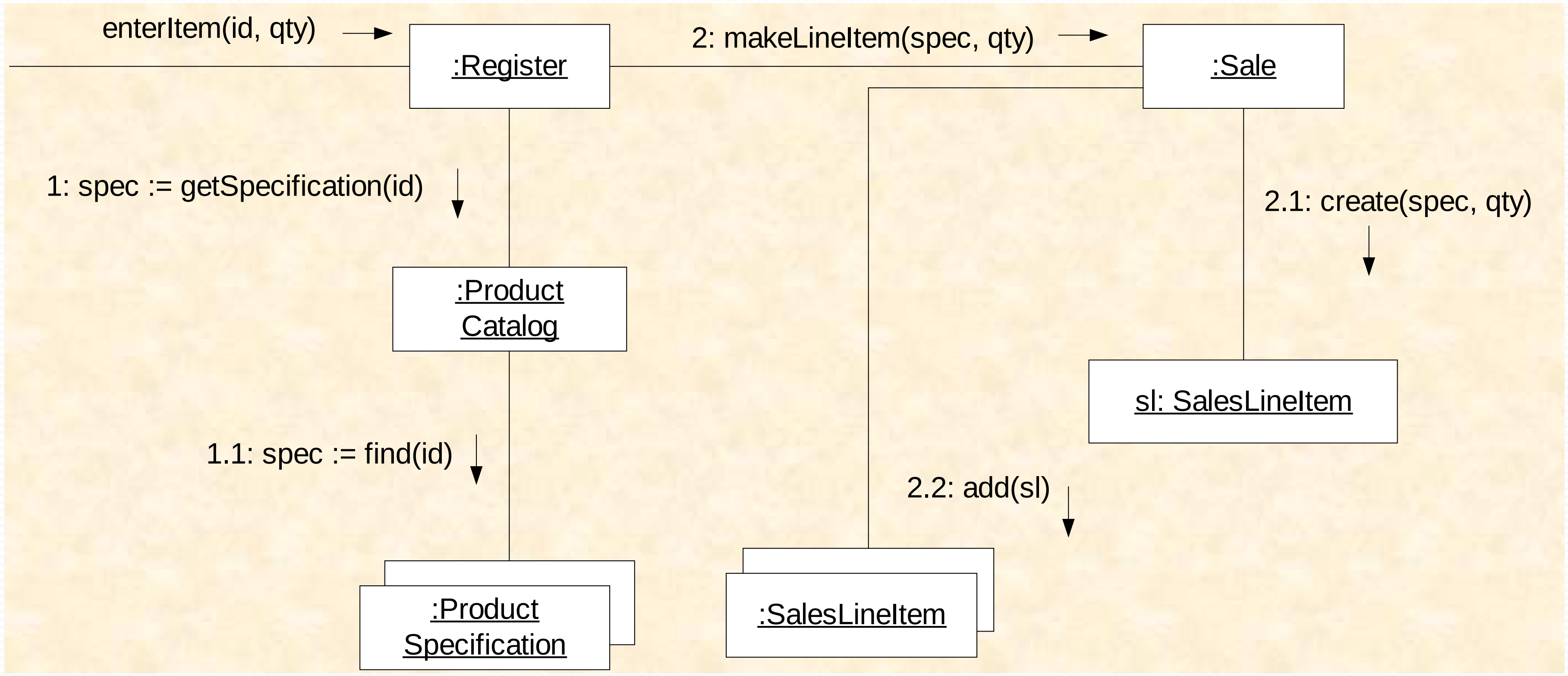
Collaboration Diagram Notations

- Illustrating instances vs. classifiers
  - Instance has designator string underlined. Note that a ":" precedes the class name.
  - A name can be used to uniquely identify the instance. Again, note that a ":" precedes the class name.



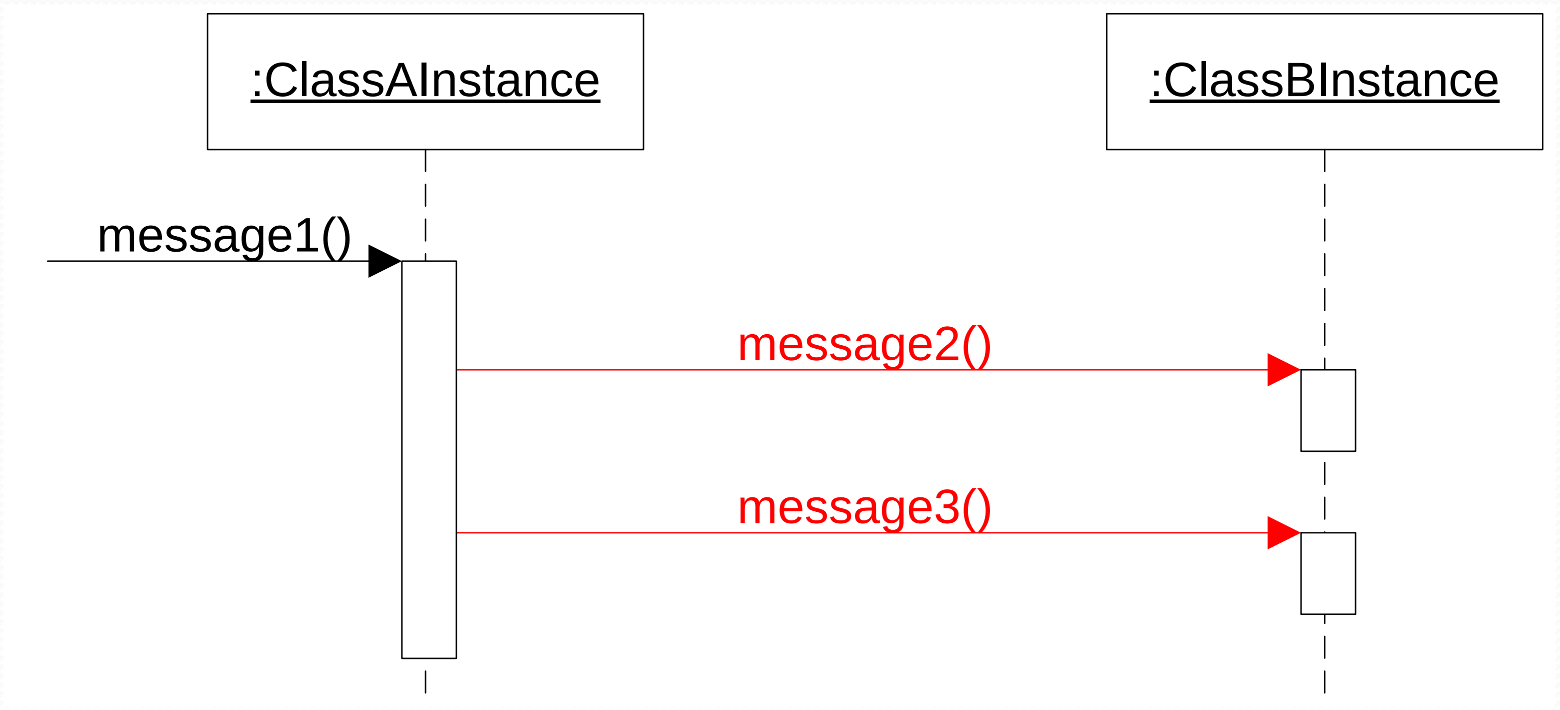
Collaboration Diagram Notations

- describe both static structure and dynamic behaviour of a system.
- do not have an explicit way to denote time
- number messages in order of execution.



Sequence Diagram

- Illustrate interactions in a kind of fence format, in which each new object is added to the right.
- Unlike Collaboration diagrams, Sequence diagrams do not show links.



UML message expression syntax

return := message(parameter : parameterType) : returnType

```
Examples:

spec := getProductSpect( id )
spec := getProductSpect( id:ItemID )
spec := getProductSpect( id:ItemID ) :
           ProductSpecification
```

Sequence vs. Collaboration Diagrams

Type	Strengths	Weaknesses
Sequence	✹ clearly shows sequence or time ordering of messages ✹ simple notation	✹ forced to extend to the right when adding new objects – consumes horizontal space
Collaboration (Communication)	✹ space economical – flexibility to add new objects in two dimensions ✹ better to illustrate complex branching, iteration, and concurrent behavior	✹ difficult to see sequence of messages ✹ more complex notation

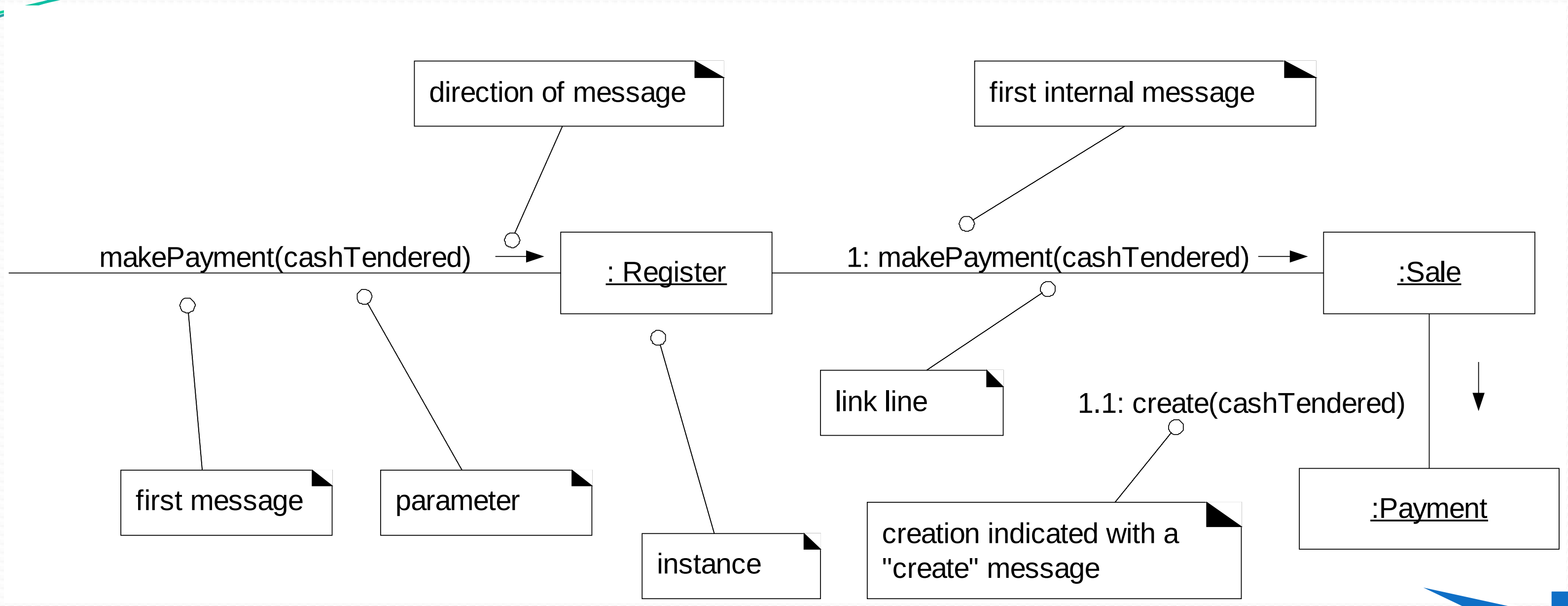
Example: makePayment

1. The message *makePayment* is sent to an instance of *Register*.

2. The *Register* instance sends the *makePayment* message to a *Sale* instance.

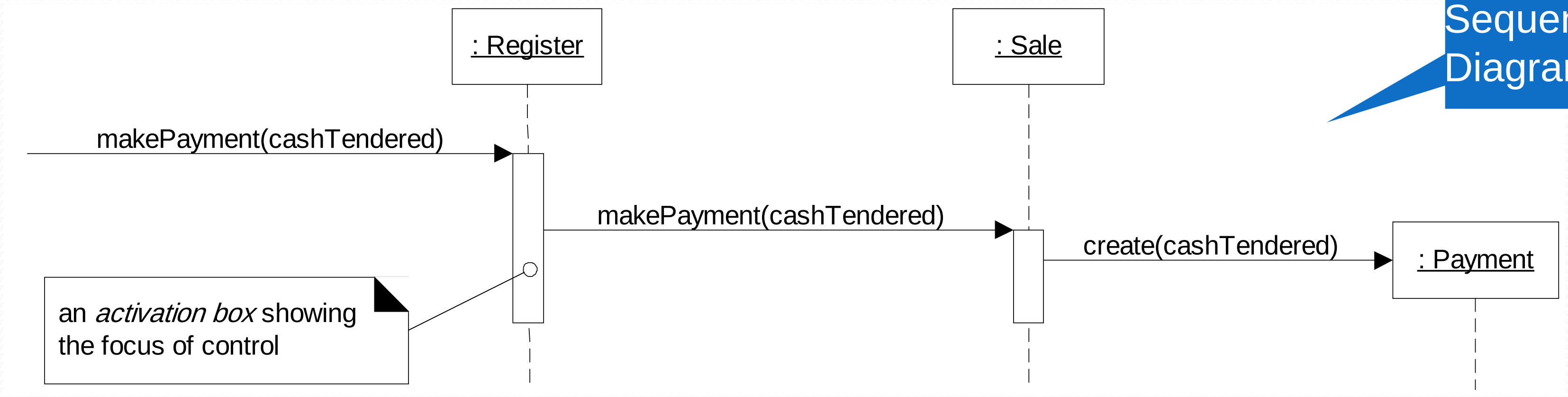
3. The *Sale* instance creates an instance of *Payment*.

Example Interaction Diagrams: makePayment


direction of message
first internal message
makePayment(cashTendered)
: Register
1: makePayment(cashTendered)
:Sale
link line
1.1: create(cashTendered)
first message
parameter
instance
creation indicated with a "create" message
:Payment

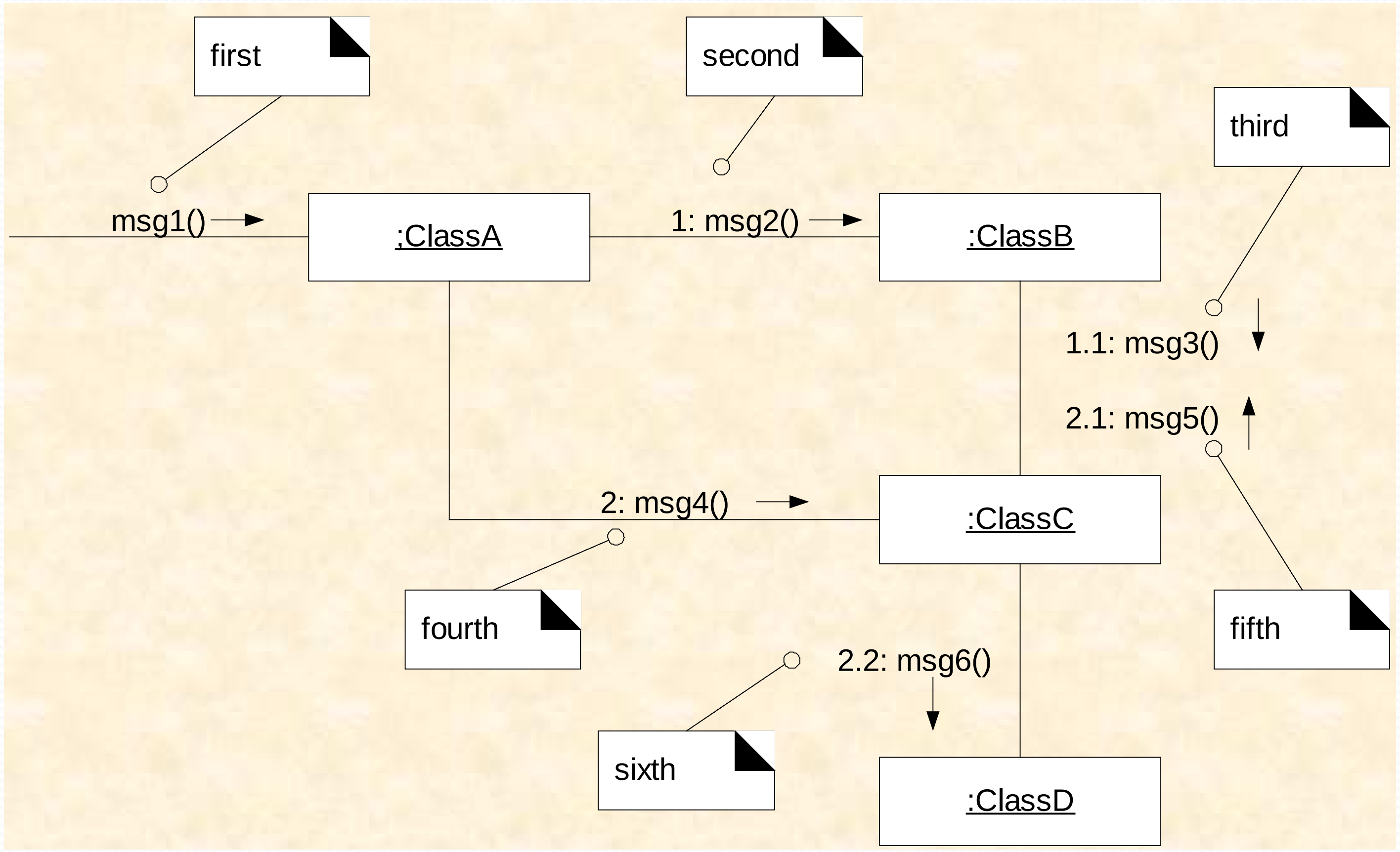
Collaboration Diagram

Sequence Diagram

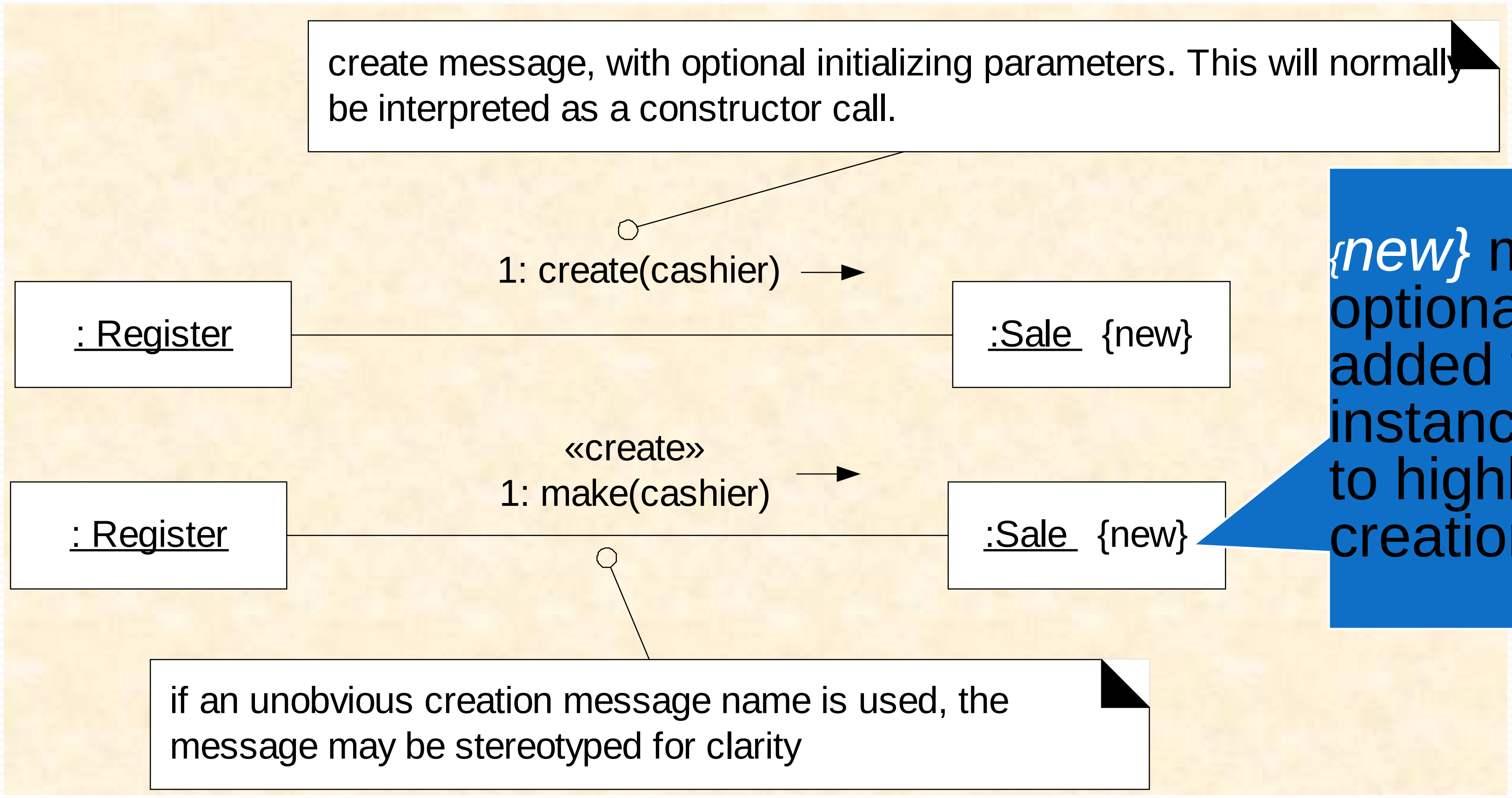

: Register
: Sale
makePayment(cashTendered)
makePayment(cashTendered)
create(cashTendered)
: Payment
an *activation box* showing the focus of control

Message numbering sequence

- Sequence numbering can become nested using legal numbering (the Dewey decimal system).
- E.g. nested messages under the first message are labeled 1.1, 1.2, 1.3, and so on.



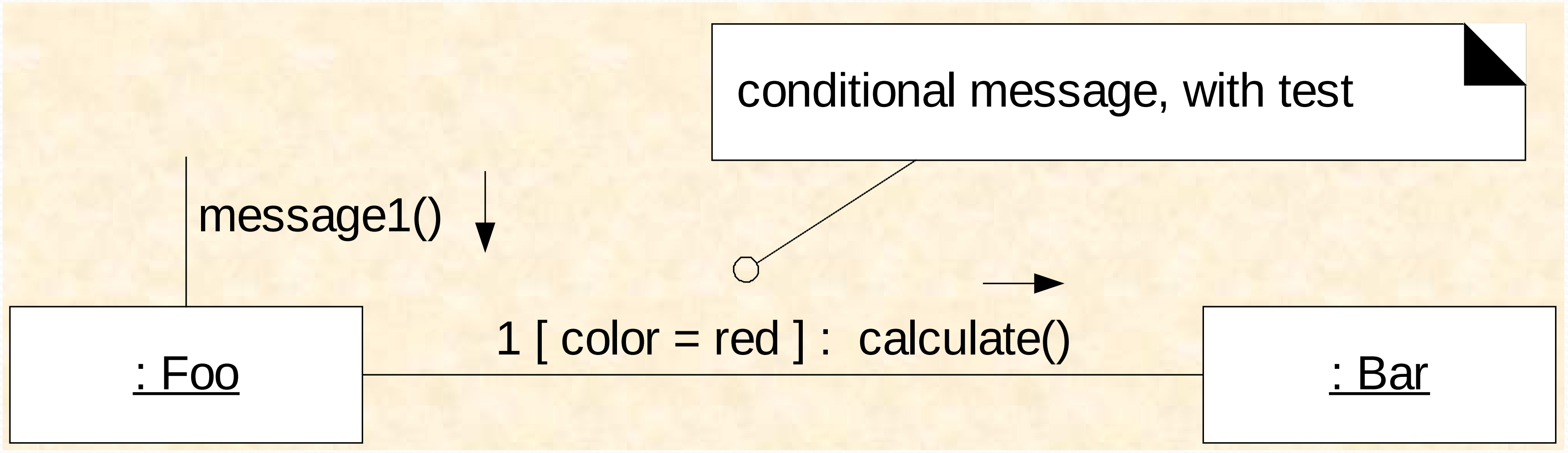
Creation of Instances


create message, with optional initializing parameters. This will normally be interpreted as a constructor call.
1: create(cashier)
: Register
:Sale {new}
«create»
1: make(cashier)
: Register
:Sale {new}
if an unobvious creation message name is used, the message may be stereotyped for clarity

{new} may be optionally added to the instance box to highlight creation

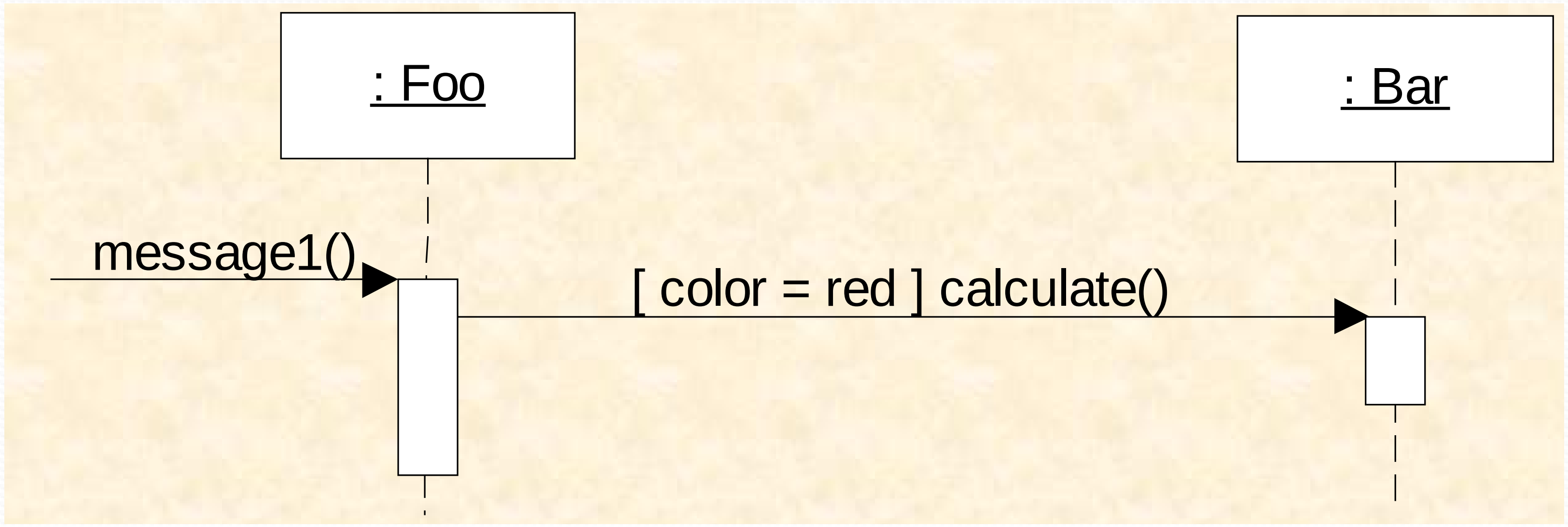
Conditional Messages - Collaboration

- The conditional message in a Collaboration Diagram is usually placed in square brackets immediately following the sequence number.
- The message is only sent if the conditional clause evaluates to true.


conditional message, with test
message1()
1 [color = red] : calculate()
: Foo
: Bar

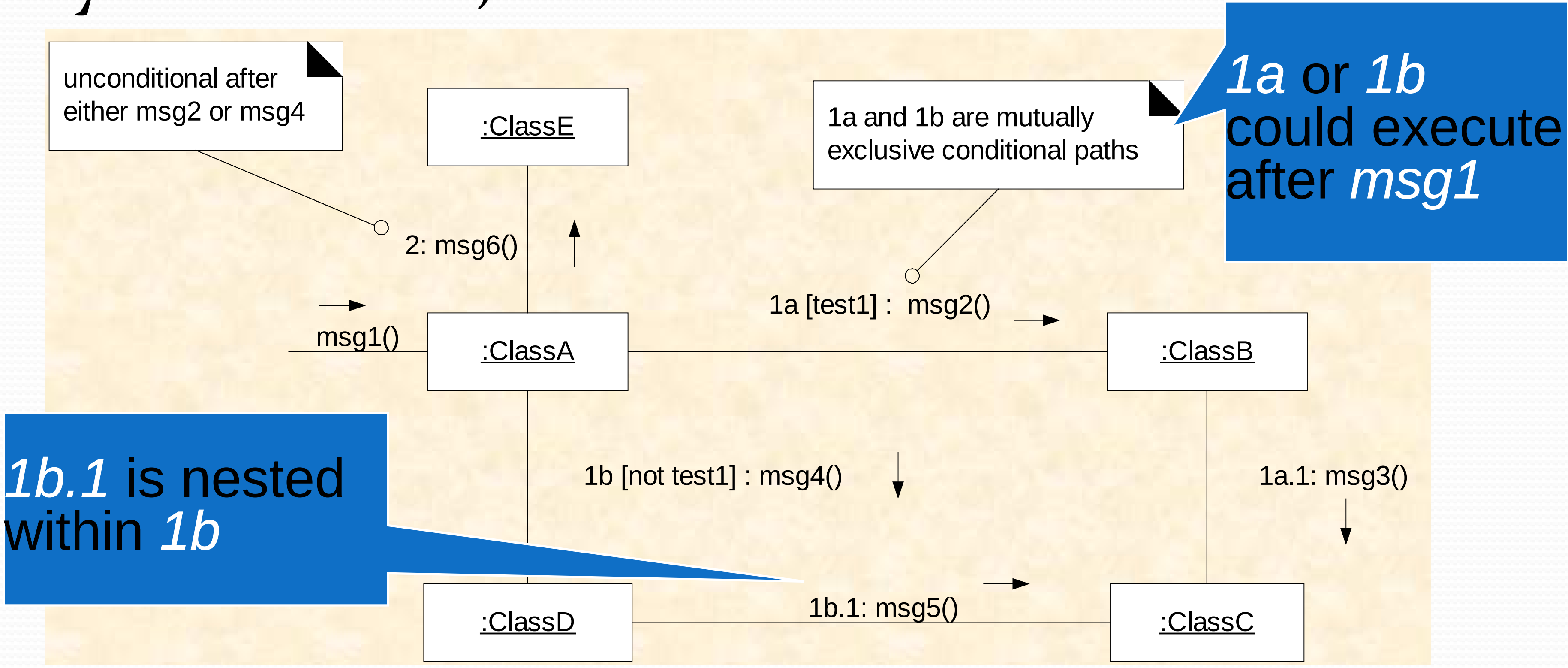
Conditional Messages – Sequence

- The condition for a message in a Sequence Diagram is also placed in square brackets



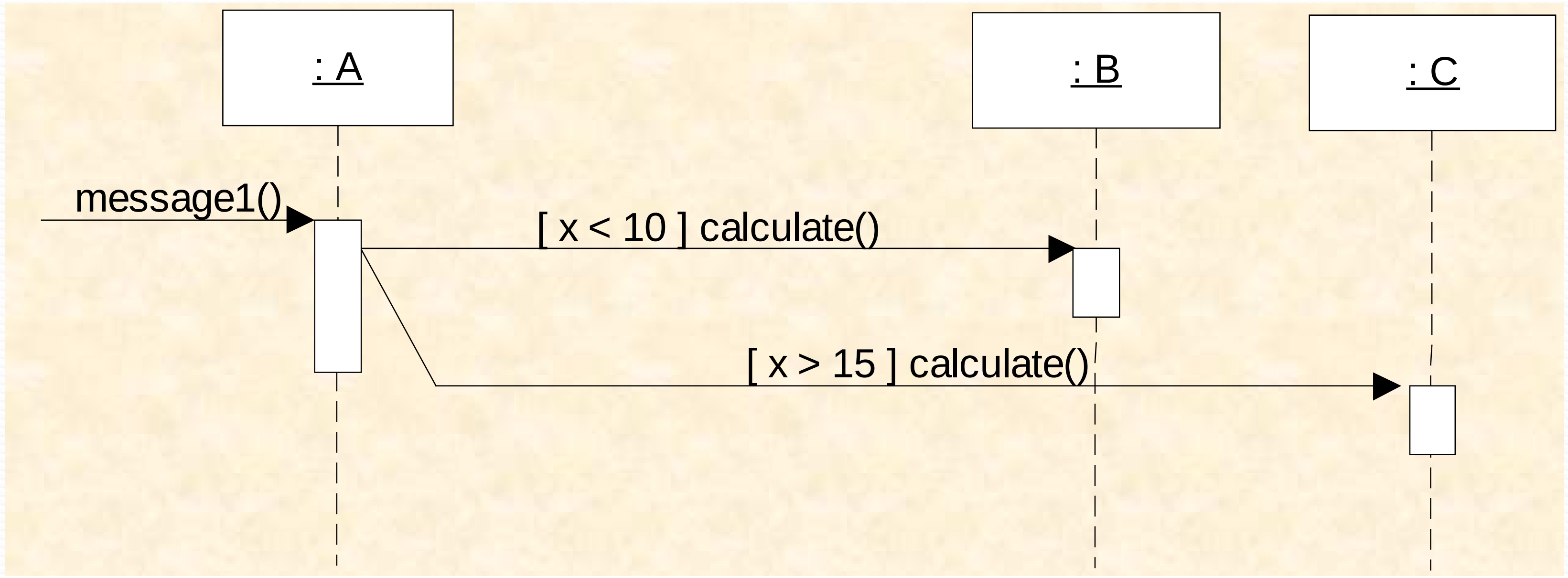
Mutually Exclusive Conditional Messages – Collabroation Diagram

- Sequence expression modified with a conditional path expression.
- By convention, the first letter used is *a*

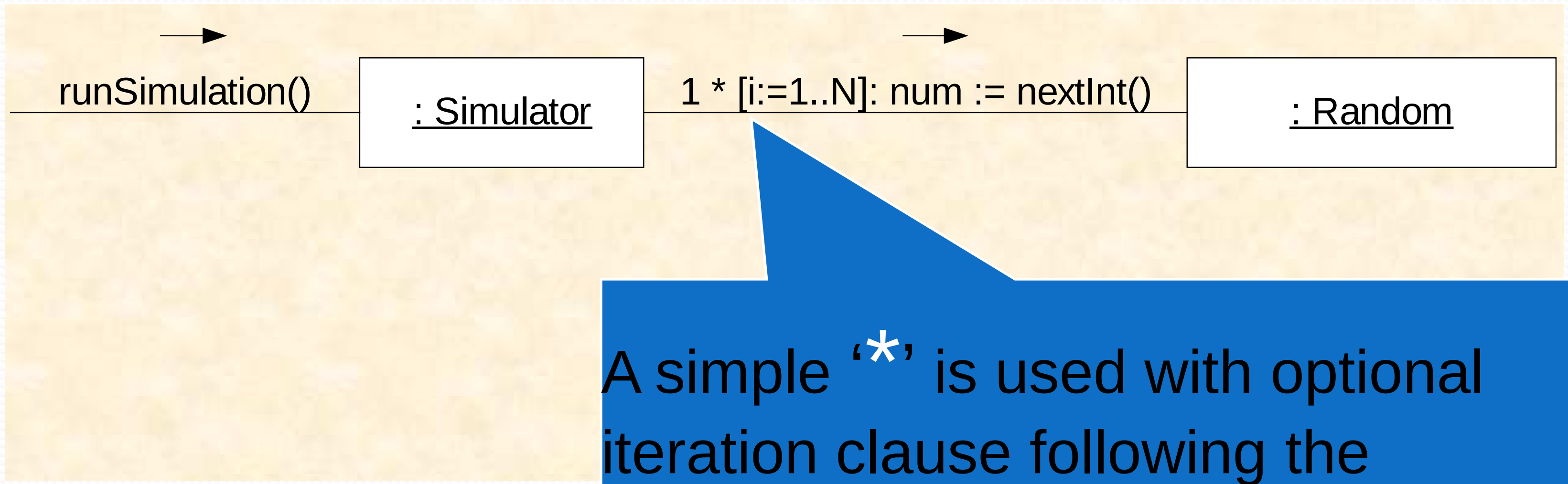

unconditional after either msg2 or msg4
:ClassE
1a and 1b are mutually exclusive conditional paths
1a or *1b* could execute after *msg1*
2: msg6()
1a [test1] : msg2()
msg1()
:ClassA
:ClassB
1b.1 is nested within *1b*
1b [not test1] : msg4()
1a.1: msg3()
1b.1: msg5()
:ClassD
:ClassC

Mutually Exclusive Conditional Messages – Sequence Diagram

- Mutually exclusive conditional messages are illustrated with a kind of angled line emerging from a common point in the Sequence Diagram.

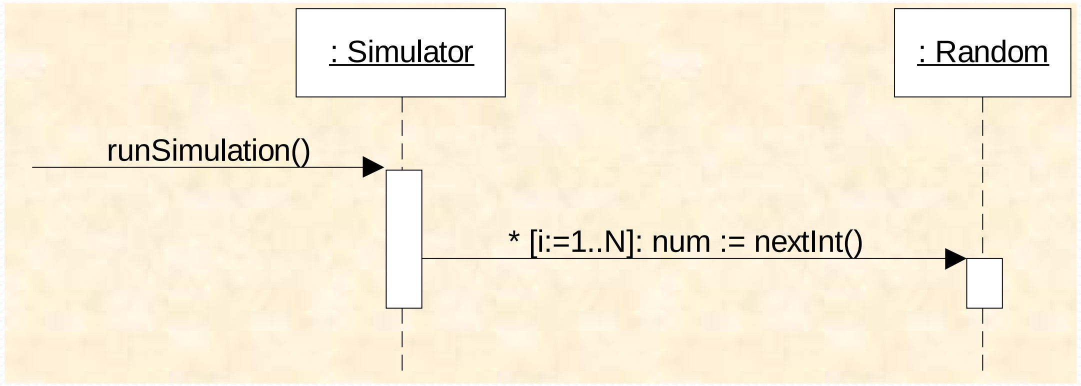


Iteration (Message Looping) – Collaboration Diagram

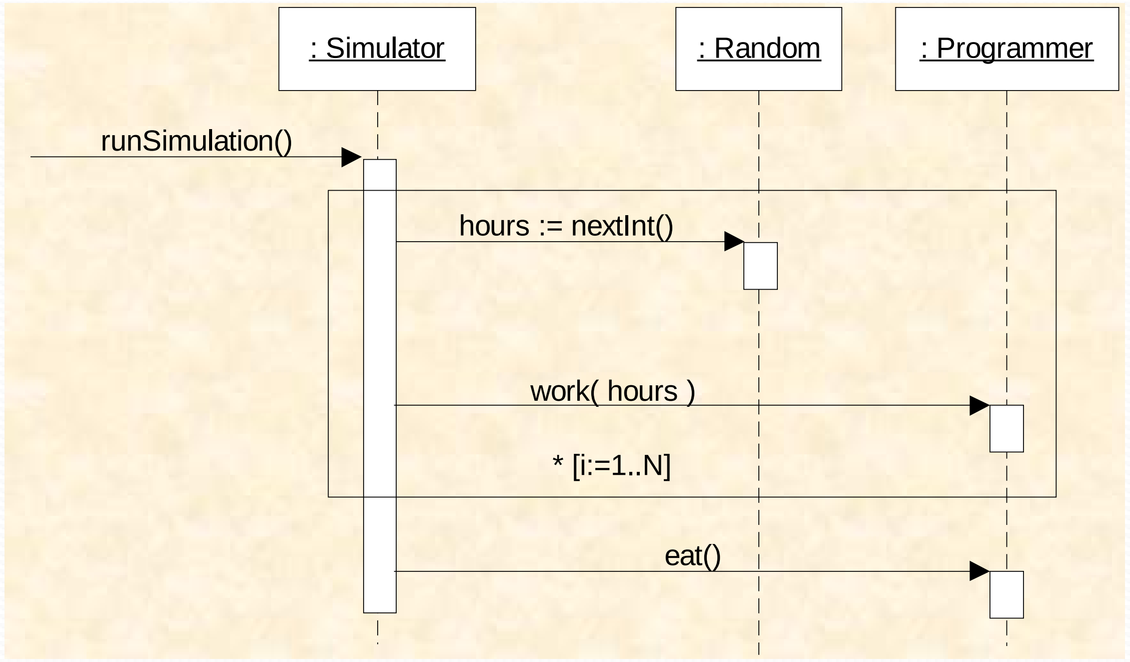


A simple '*' is used with optional iteration clause following the sequence number

Iteration (Message Looping) – Sequence Diagram

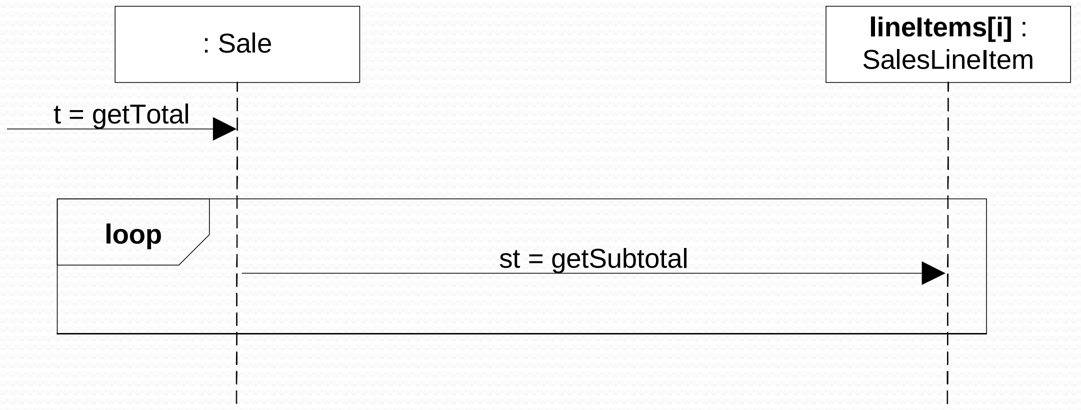


Single message iteration

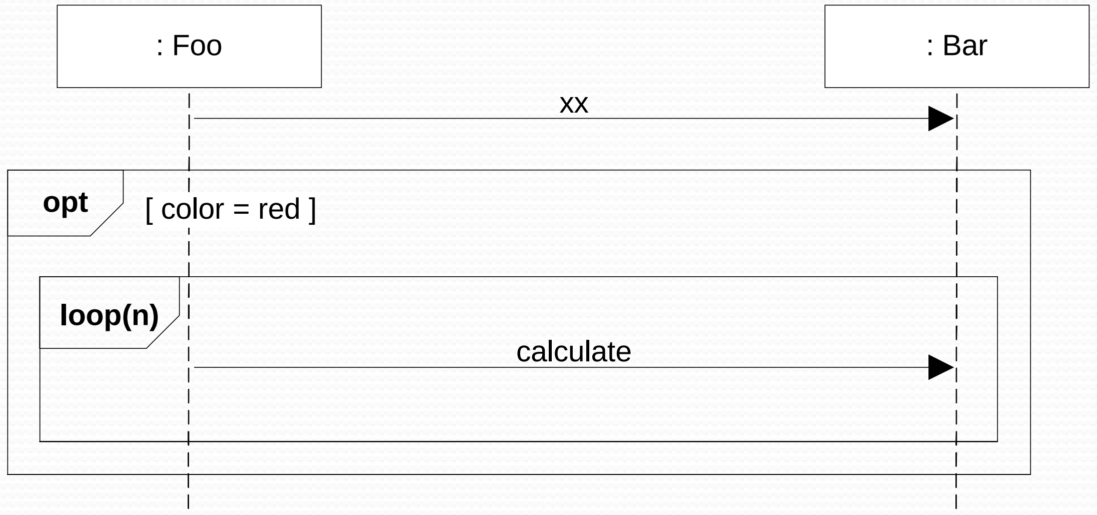


Iteration of series of messages

Iteration (Message Looping) – Nesting Interaction Frames

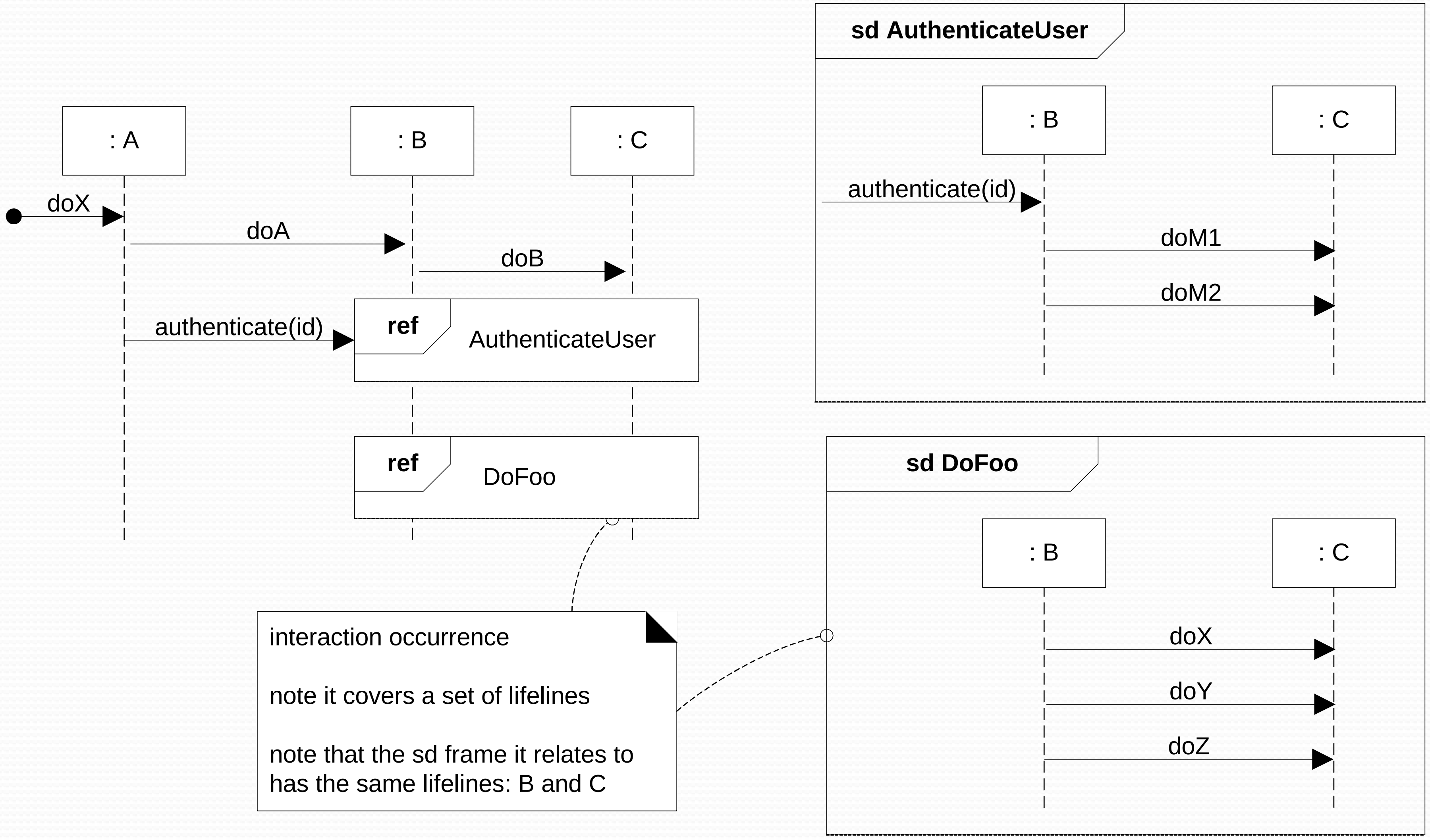


Single message Interaction Frame



Iteration Nesting Interaction Frames

Reference Interaction Frames

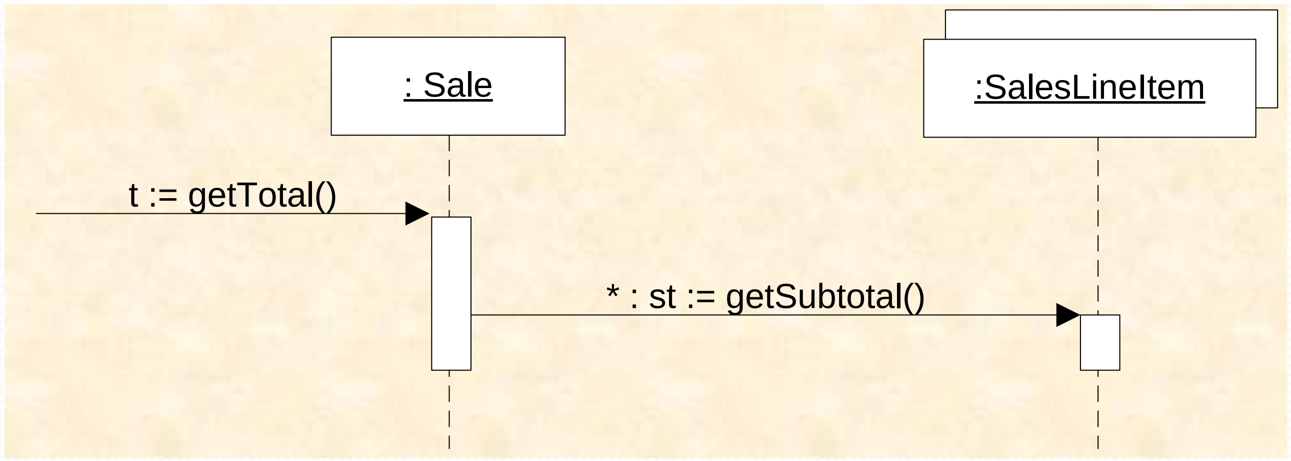


interaction occurrence

note it covers a set of lifelines

note that the sd frame it relates to has the same lifelines: B and C

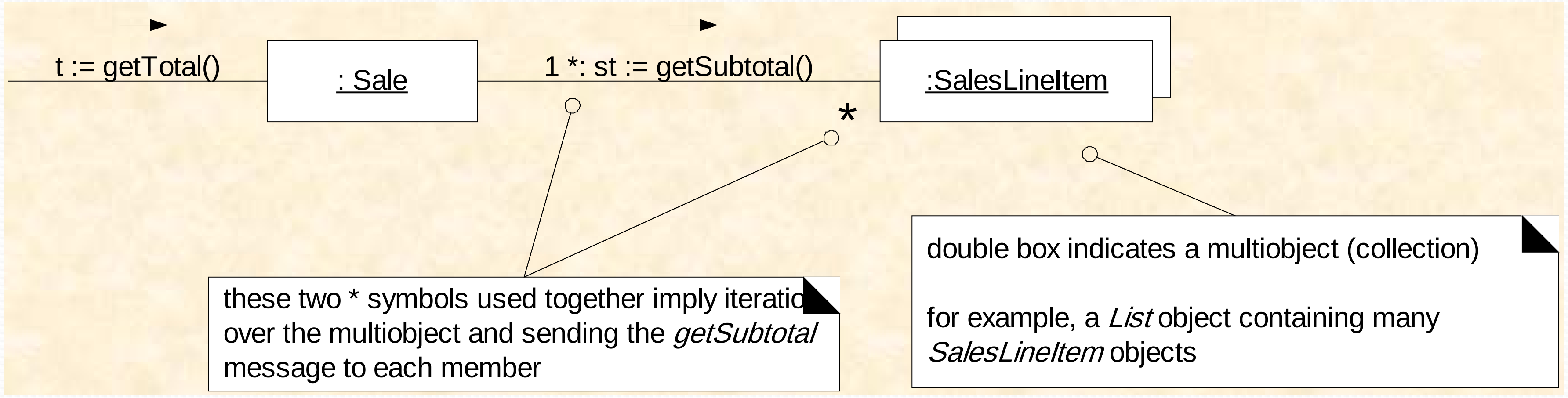
Iteration (Looping) Over A Collection – Sequence Diagram



Iteration over a collection

Iteration (Looping) Over A Collection – Collaboration Diagram

- UML term *multiobject* is used to denote collection
- A '*' multiplicity marker at end of link is used to indicate that message is being sent to each element of collection



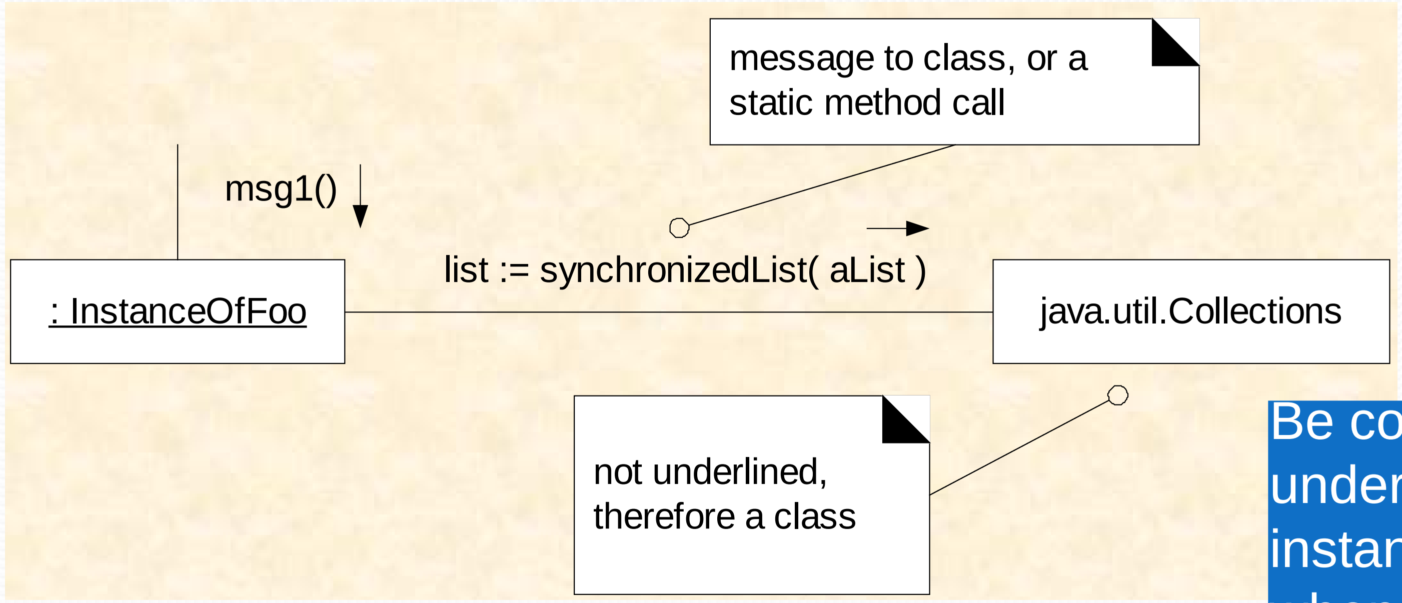
these two * symbols used together imply iteratio over the multiobject and sending the *getSubtotal* message to each member

double box indicates a multiobject (collection)

for example, a *List* object containing many *SalesLineItem* objects

Messages To A Class Object – Collaboration Diagram

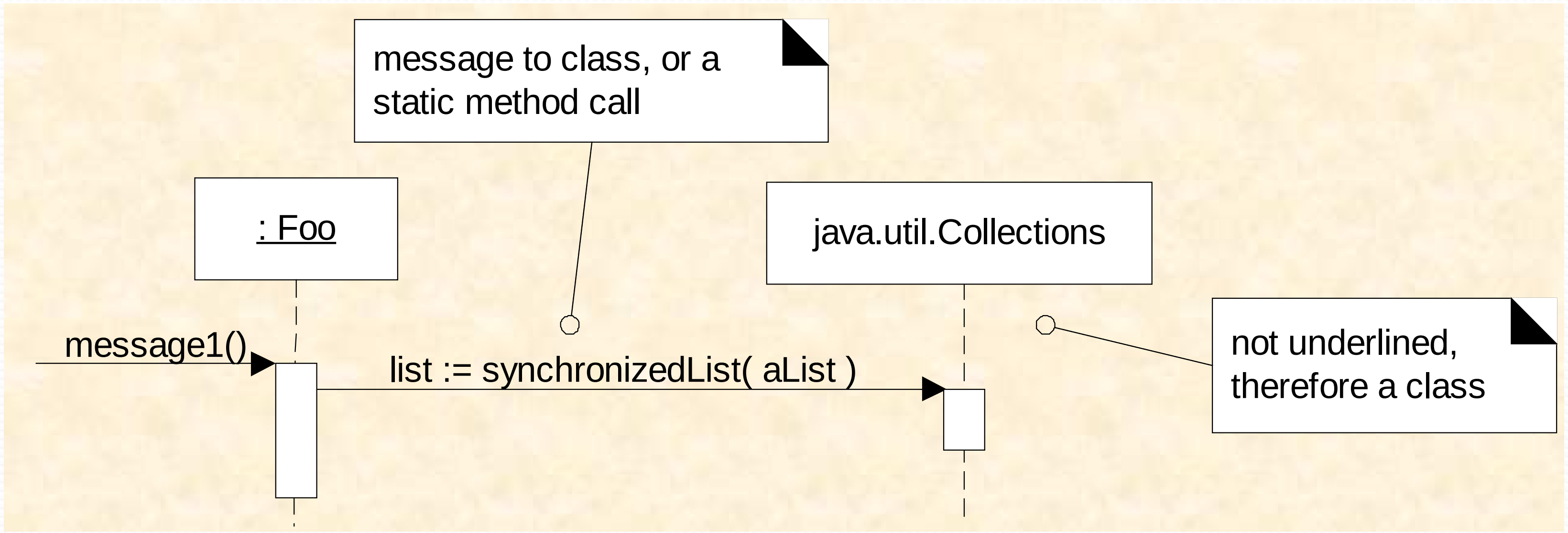
- A Class Object contains static methods.
- This is shown with name not underlined, indicating that message is being sent to a class rather than an instance


message to class, or a static method call
msg1()
list := synchronizedList(aList)
: InstanceOfFoo
java.util.Collections
not underlined, therefore a class

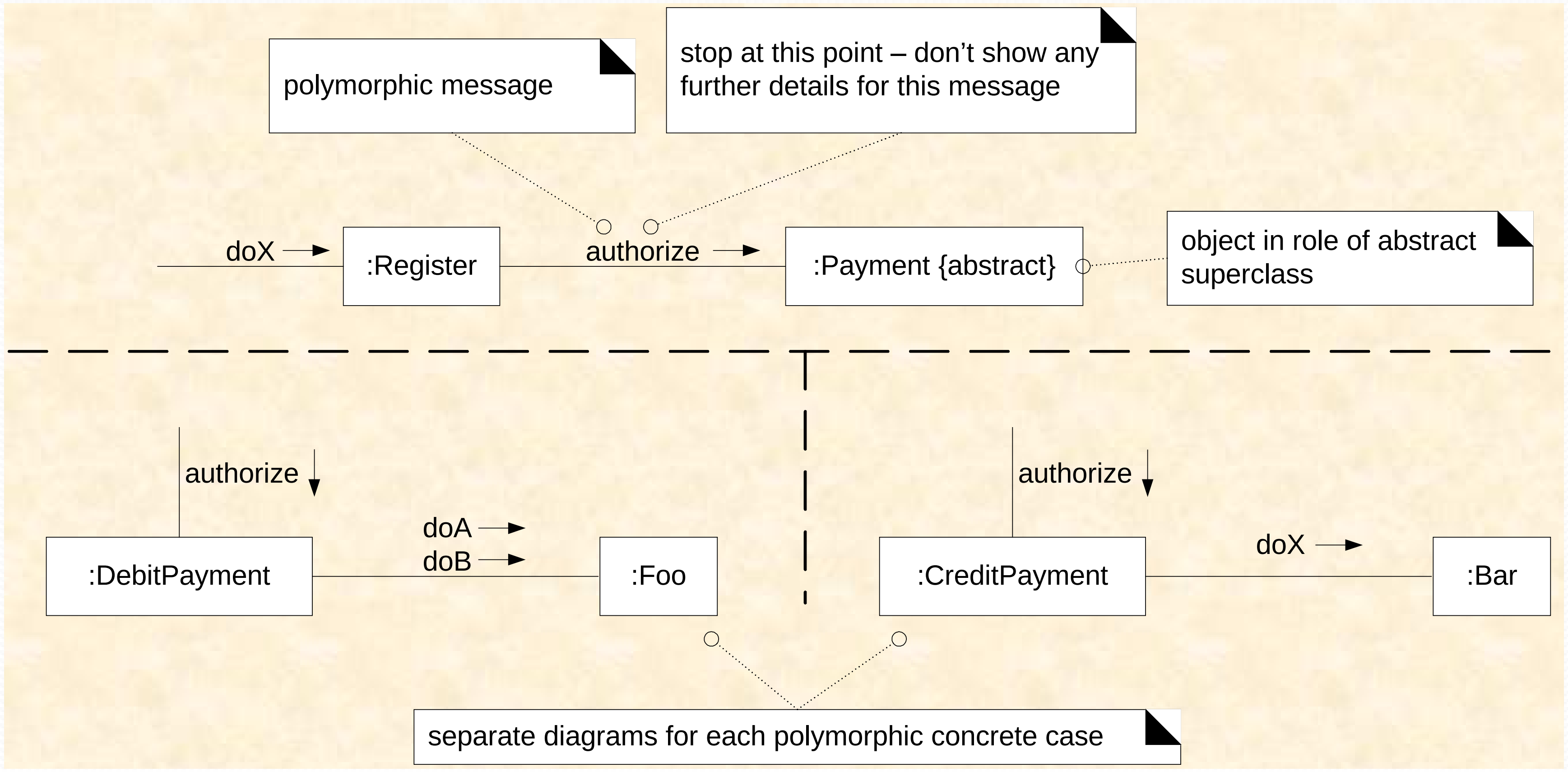
Be consistent in underlining instance names when instance is intended.

Messages To A Class Object – Sequence Diagram

- Class or static method calls are shown by not underlining the name of the classifier, which signifies a class object rather than an instance.


message to class, or a static method call
: Foo
java.util.Collections
message1()
list := synchronizedList(aList)
not underlined, therefore a class

Polymorphic Messages – Collaboration Diagram


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

Dynamic Object Modeling

- *Guideline*
  - *Spend significant time doing interaction diagrams (sequence or communication diagrams), not just static object modeling with class diagrams.*
  - *Ignoring this guideline is a very common worst-practice with UML*

Quote from Larman, p.217