Design: Terminology and Notations

COMP 3700.002
Software Modeling and Design

Shehenaz Shaik

Misc…

- Terminology
- Concepts
- Notations

Actors

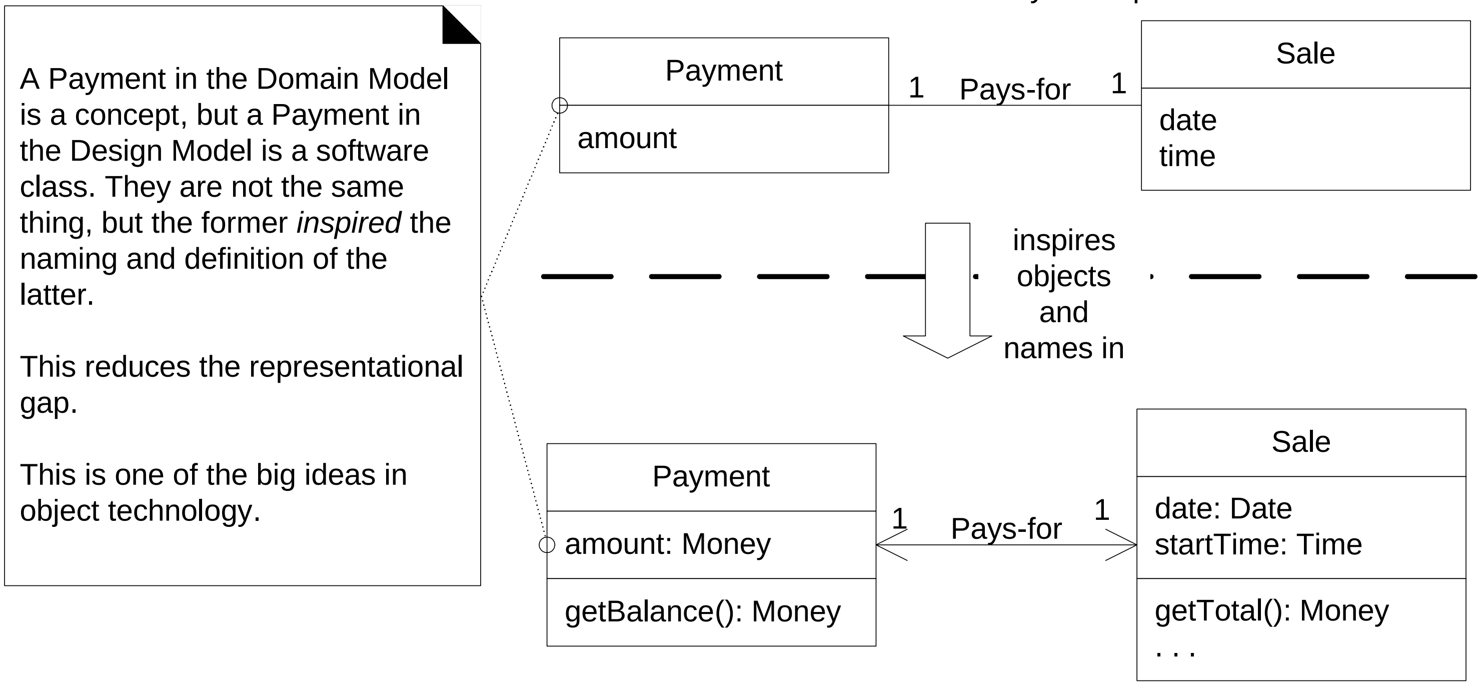
- Primary actor
  - Has user goals fulfilled through using services of the SuD
- Supporting actor
  - Provides a service to the SuD
- Offstage actor
  - Has an interest in the behavior of use case

SuD – System Under Design

Low Representational Gap (LRG)

1010101011100101101101001010010010101111

UP Domain Model
Stakeholder's view of the noteworthy concepts in the domain.



A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

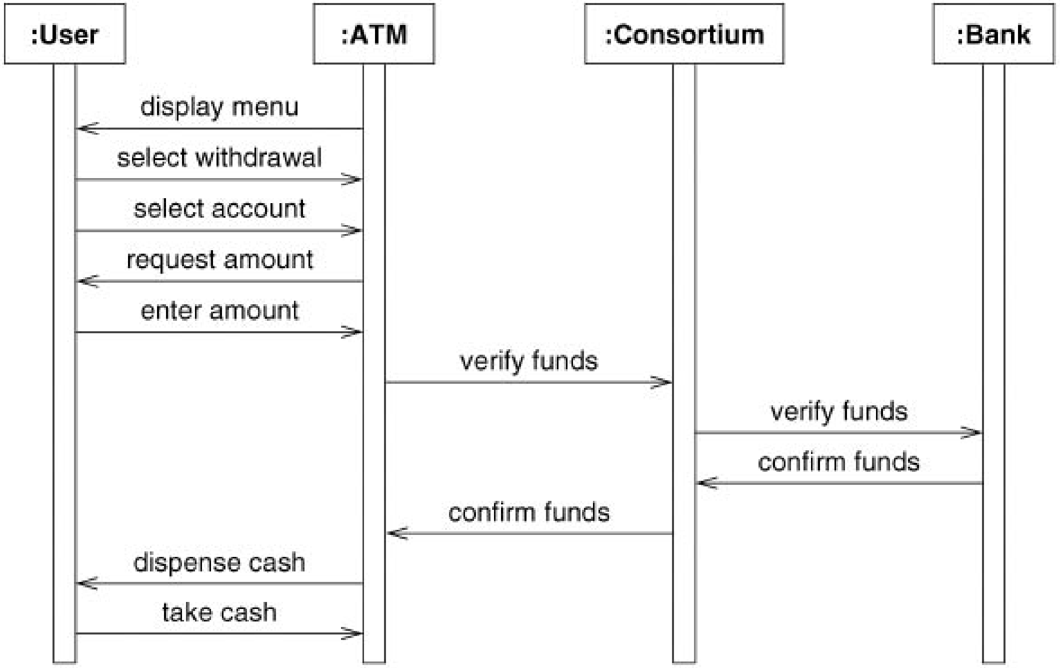
This is one of the big ideas in object technology.

UP Design Model
The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

ATM: System Sequence Diagram(SSD)

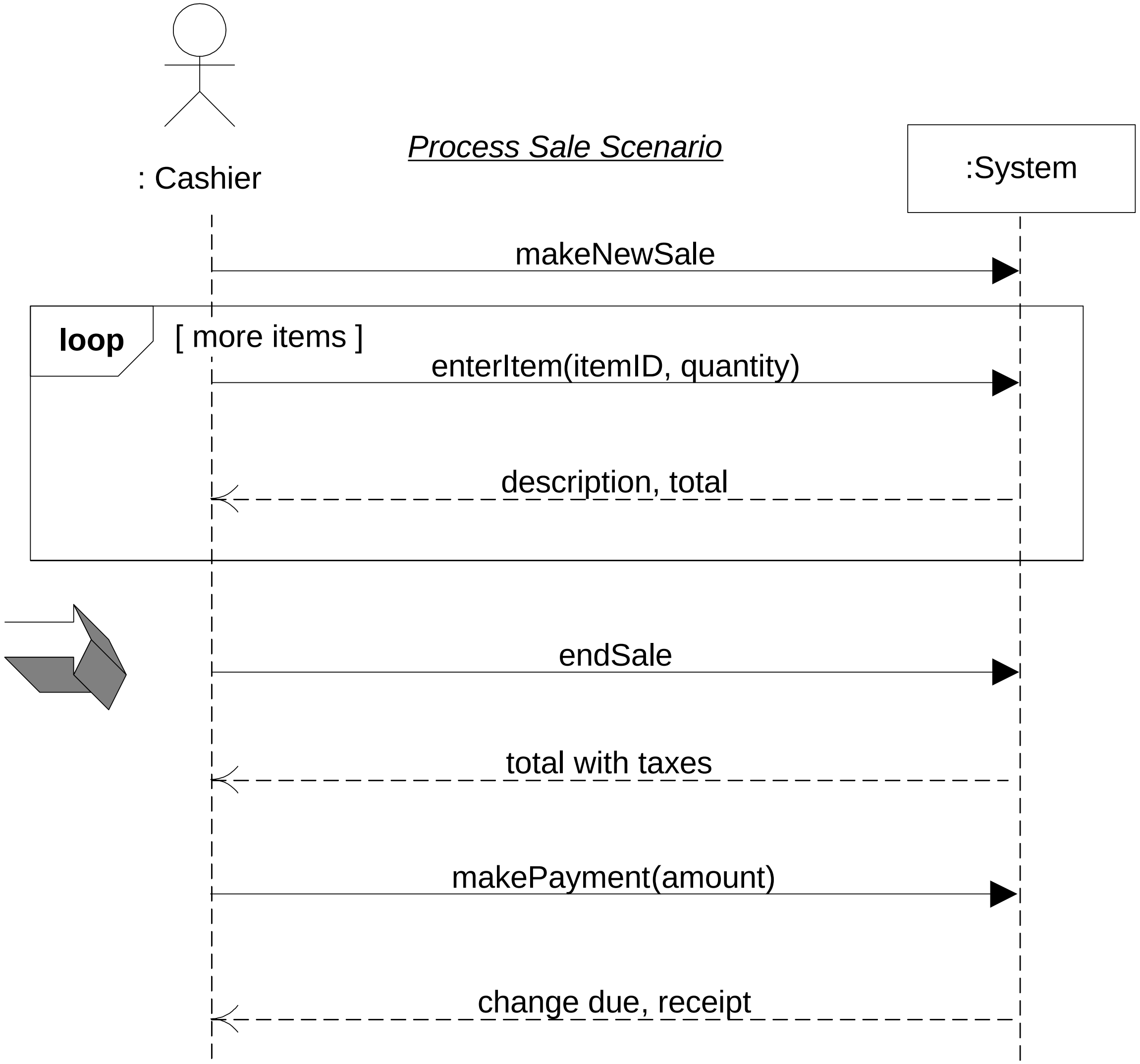
- System events
  - External events from actors
  - Timer events
  - Faults / exceptions (from external sources)



Process Sale scenario: SSD

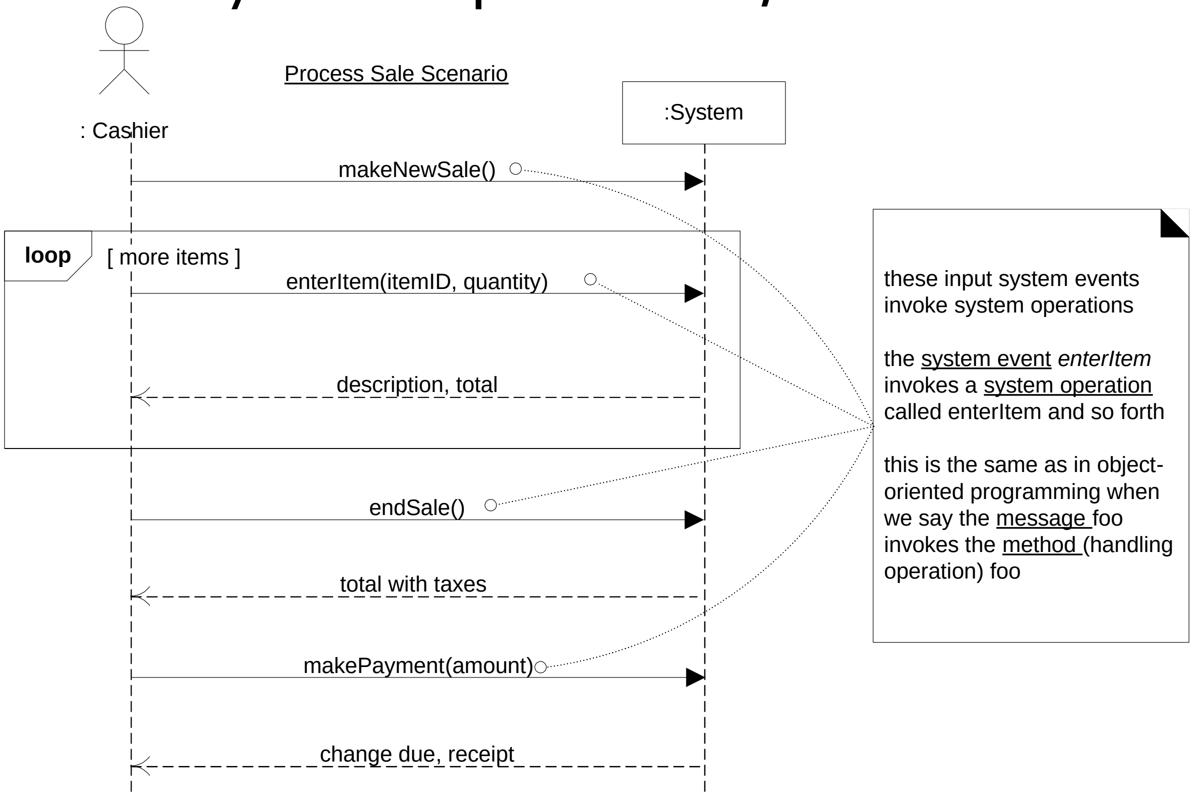
Simple cash-only *Process Sale* scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...



System Interface

- System operation
- Public System interface
  - Set of all system operations, across all use cases



these input system events invoke system operations

the system event *enterItem* invokes a system operation called enterItem and so forth

this is the same as in object-oriented programming when we say the message foo invokes the method (handling operation) foo

Operation Contract

- Describes detailed changes to objects in a domain model, resulting from system operation
- Applicable to system operations
    - And, operations on other components as well

- Defines
  - Name
  - Cross references
    - Applicable use cases
  - Preconditions
    - Assumptions about state of system/objects
  - Postconditions
    - State of objects after completion of operation

Operation Contract (Contd.)

- Postconditions - categories
  - Instance creation and deletion
  - Attribute change of value
  - Associations (Links) formed and broken

- Note
  - Postconditions applicable to domain objects only, not software objects from design model

Operation Contract: Example

- Operation: enterItem(itemID : ItemID, quantity : integer)
- Cross References: Use Cases: Process Sale
- Preconditions: There is a sale underway.
- Postconditions:
  - A SalesLineItem instance sli was created (instance creation).
  - sli was associated with the current Sale (association formed).
  - sli.quantity became quantity (attribute modification).
  - sli was associated with a ProductSpecification, based on itemID match (association formed).

Operation Contract:
How to create and write contracts

- Identify system operations from the SSDs
- For system operations that are complex / subtle in their results / not clear in the use case, construct a contract
- To describe the postconditions, use the following categories:
  - Instance creation and deletion
  - Attribute modification
  - Associations formed and broken

Operation and Operation Contract

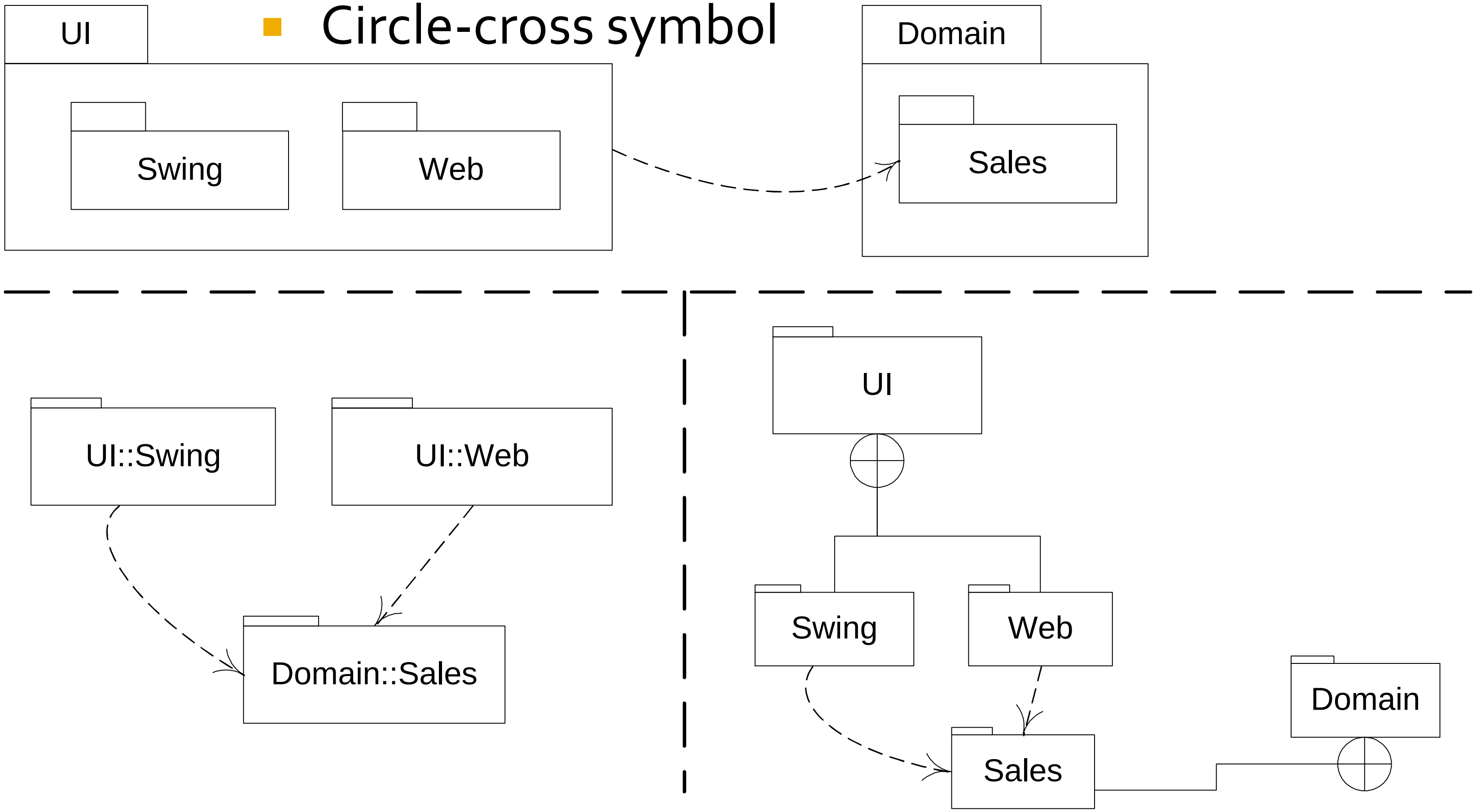
- Operation has
  - Signature (name, parameters)
  - Set of constraint objects
    - Preconditions
    - Postconditions
    - Specify semantics of operation
- Operation constraints are represented as operation contracts

Package diagram

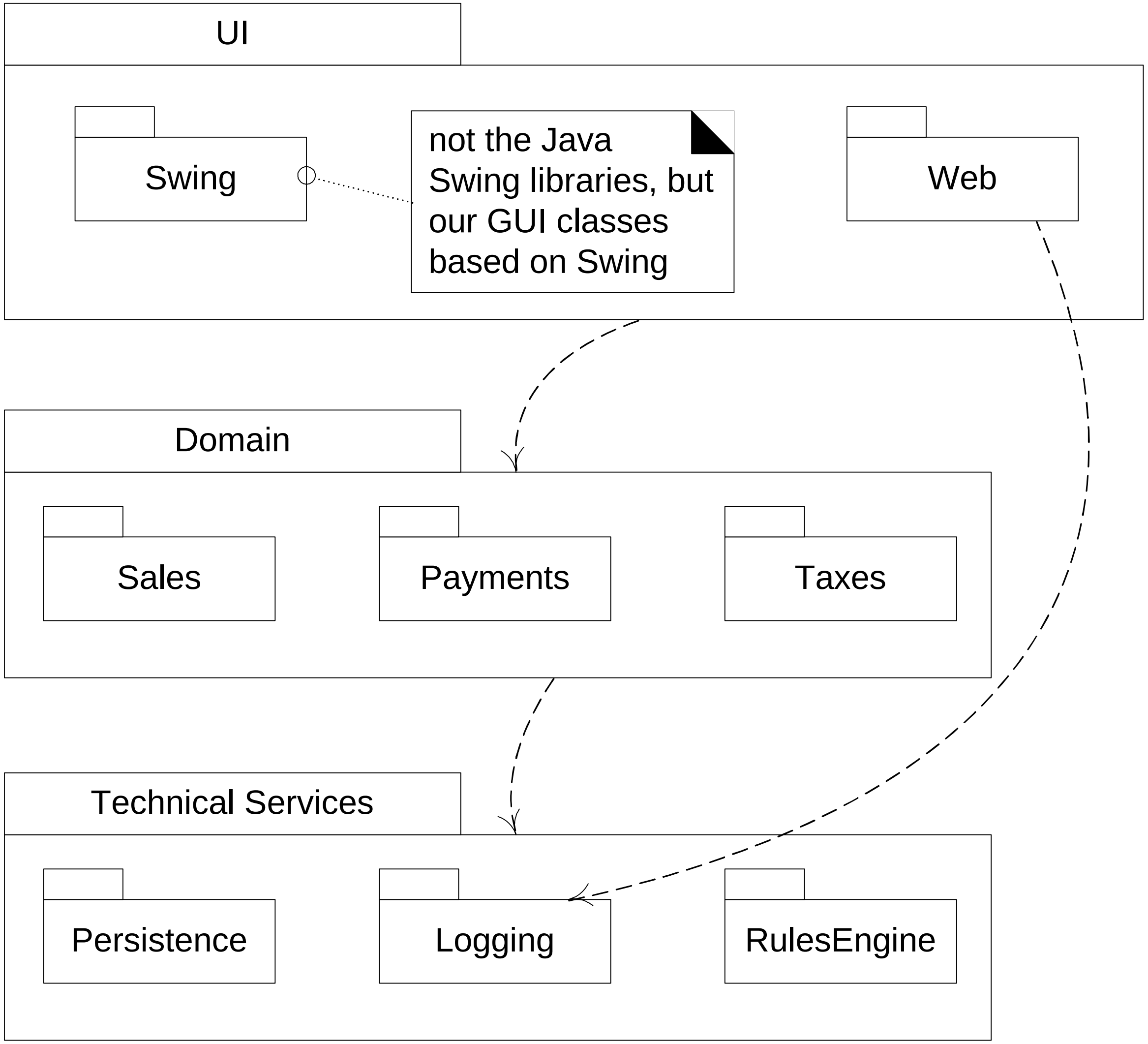
- Logical architecture of a system
  - Layers
  - Subsystems
  - Packages
- Dependency line
- Namespace

Package diagram: Notation

- Dependency
- Package nesting
- Fully qualified names
- Circle-cross symbol



Package diagram: Example



Domain layer

- Analysis model → Real-world objects
- Design model → Software objects
  - Names / Information similar to real-world objects
  - Assign application-logic responsibilities to them
  - Goal: LRG
  - Domain objects
- Domain Layer
  - Comprises domain objects to handle application logic
  - Same as Application Logic Layer

Domain Layer – Domain Model relationship

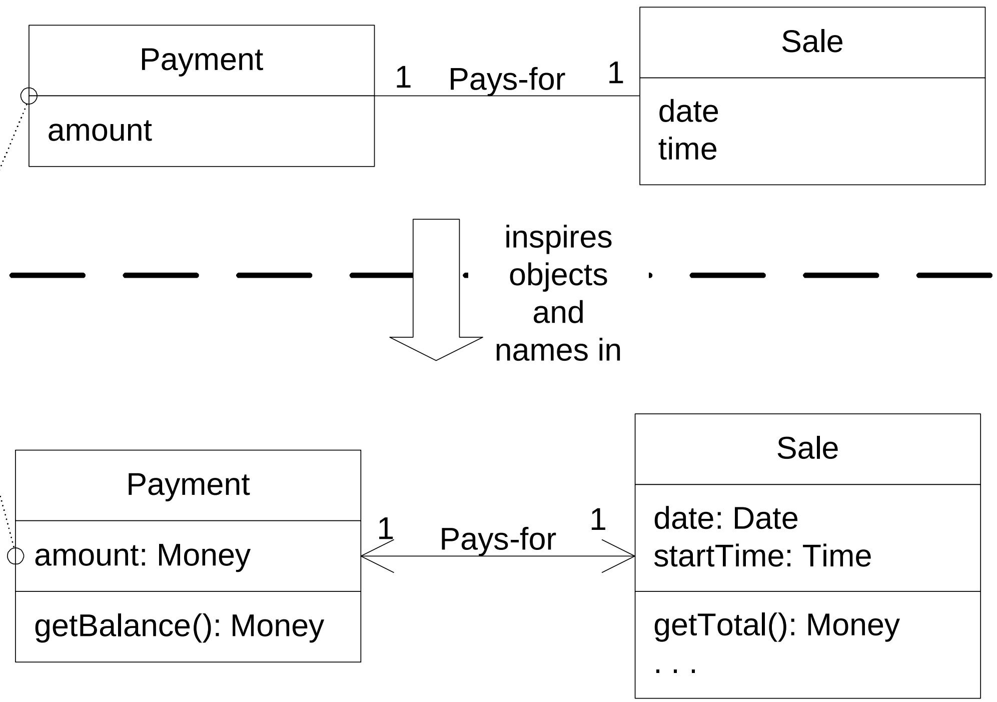
A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former inspired the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

UP Domain Model

Stakeholder's view of the noteworthy concepts in the domain.

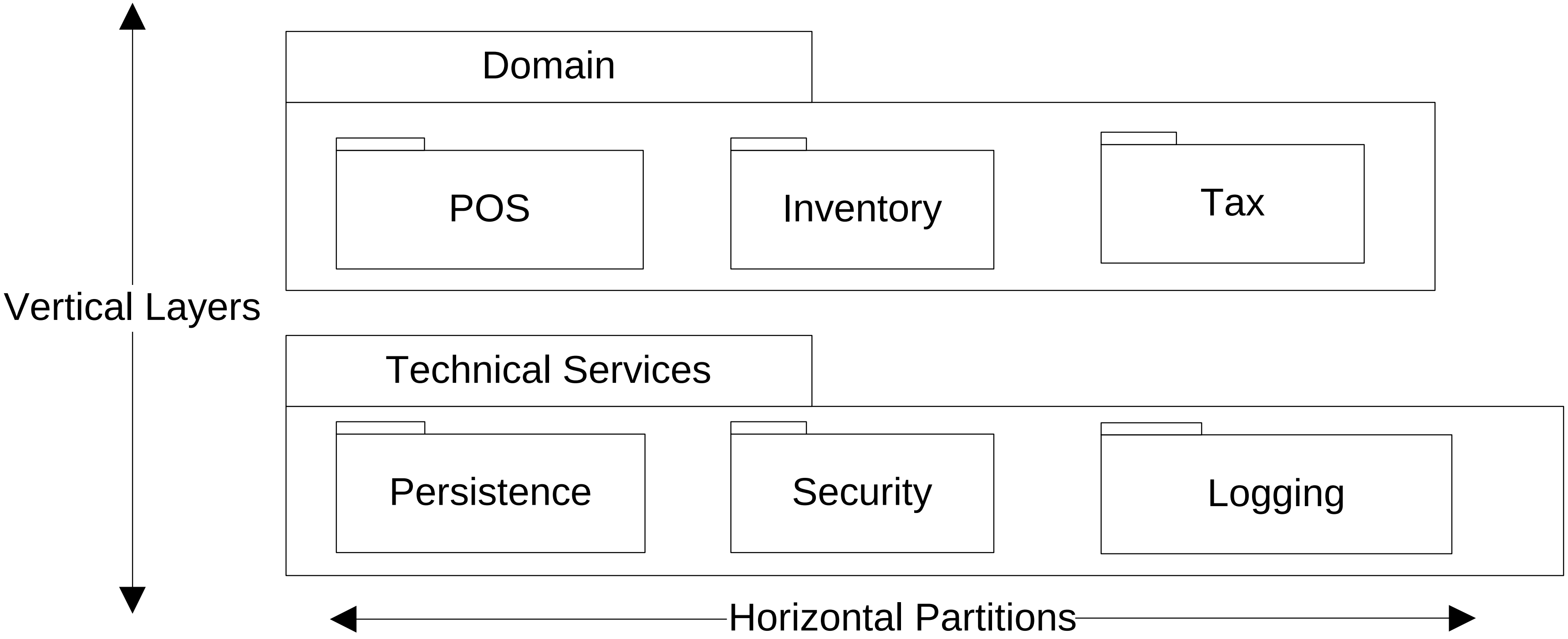


Domain layer of the architecture in the UP Design Model

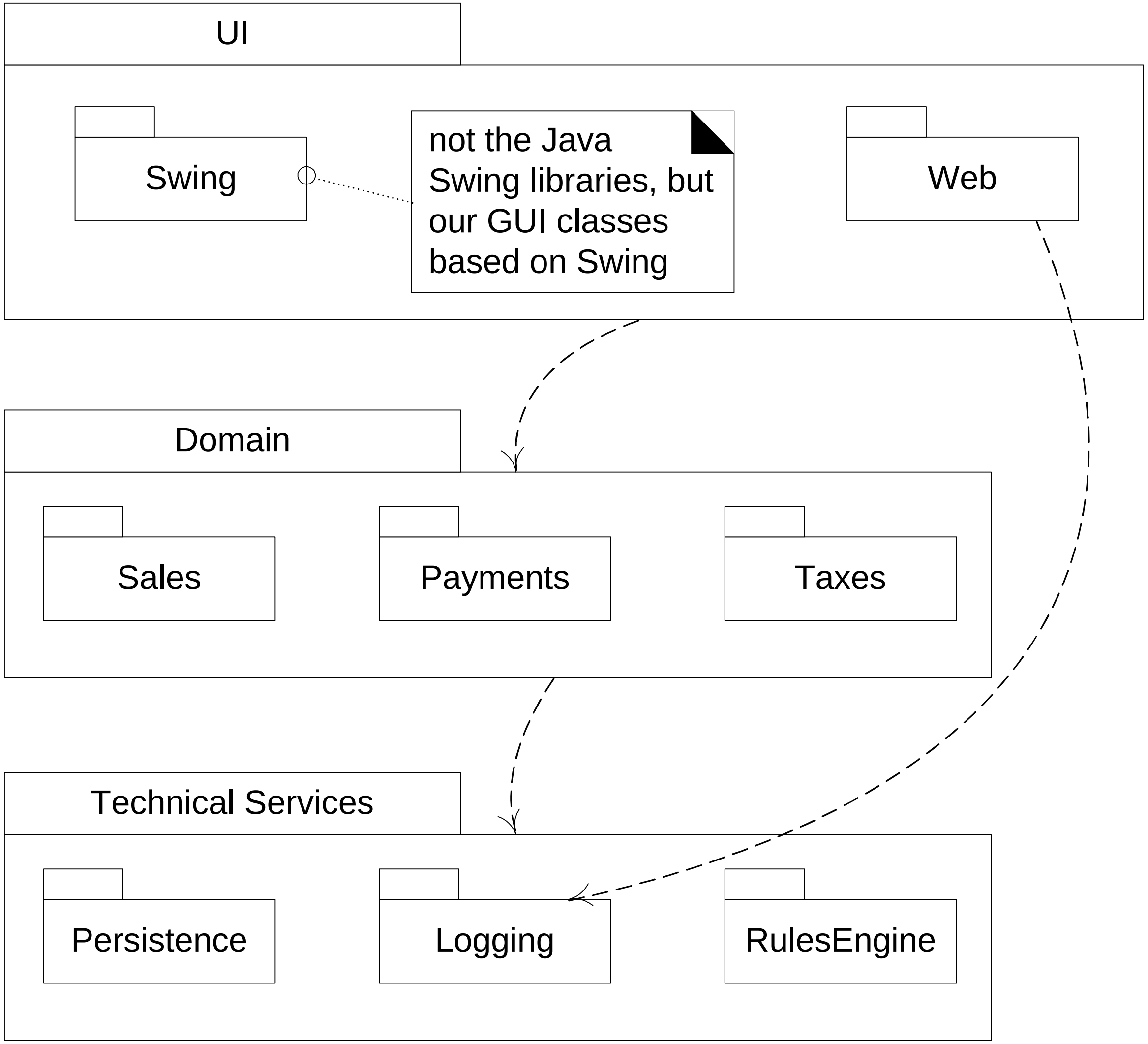
The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

Layers and Partitions

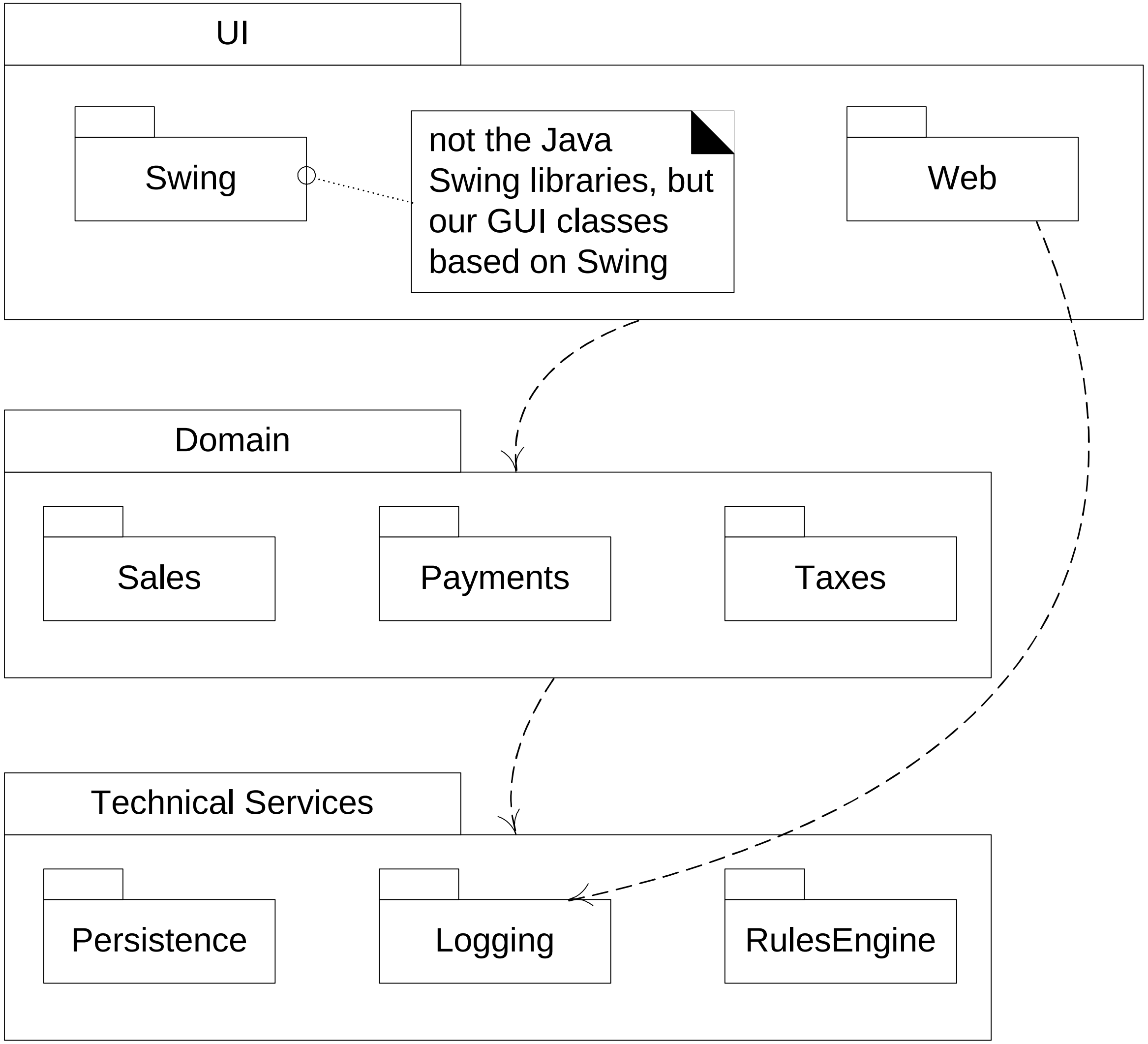


Domain Layer

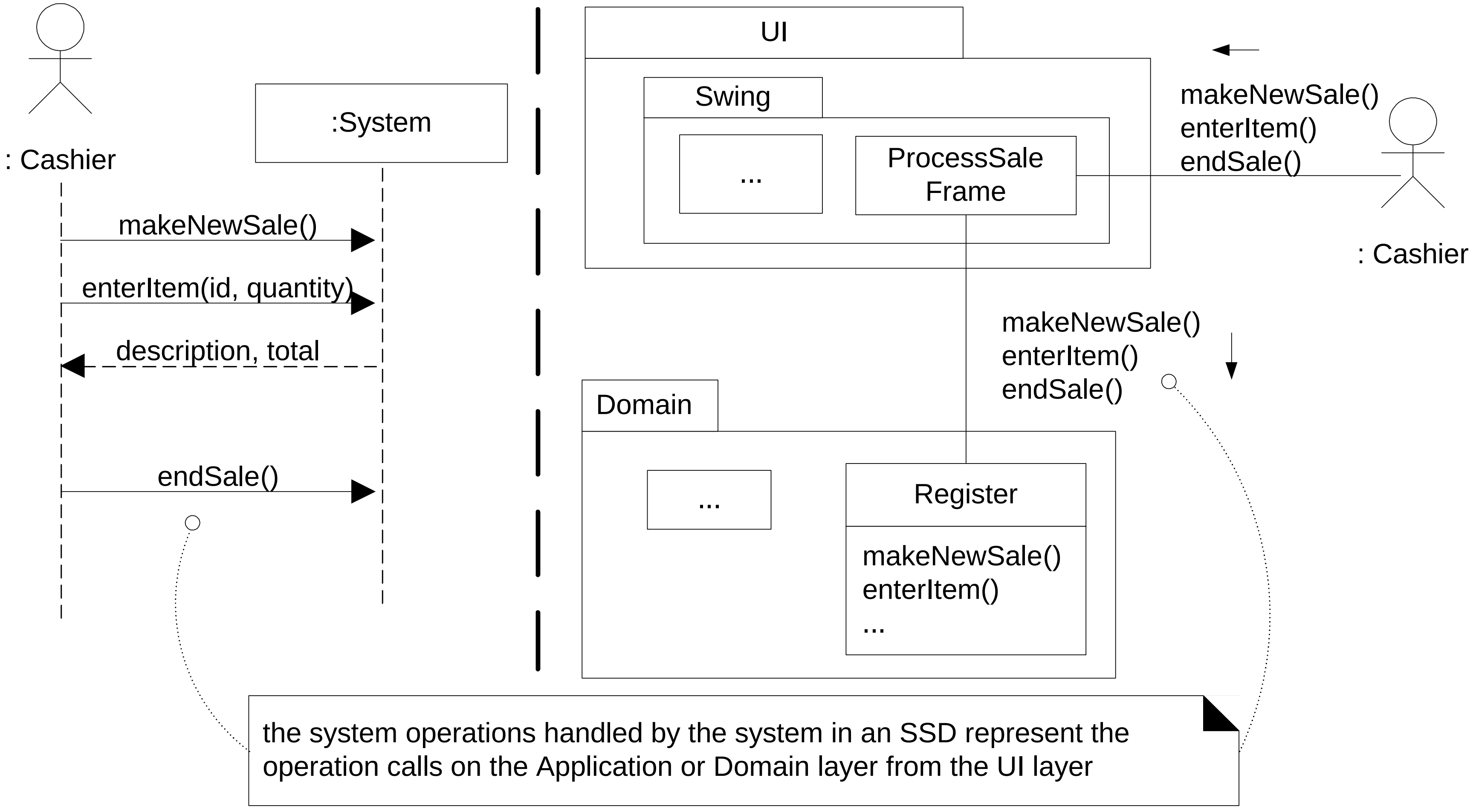


not the Java Swing libraries, but our GUI classes based on Swing

Domain Layer Focus



SSDs, System operations, Layers: Connection



the system operations handled by the system in an SSD represent the operation calls on the Application or Domain layer from the UI layer

Interaction Diagrams

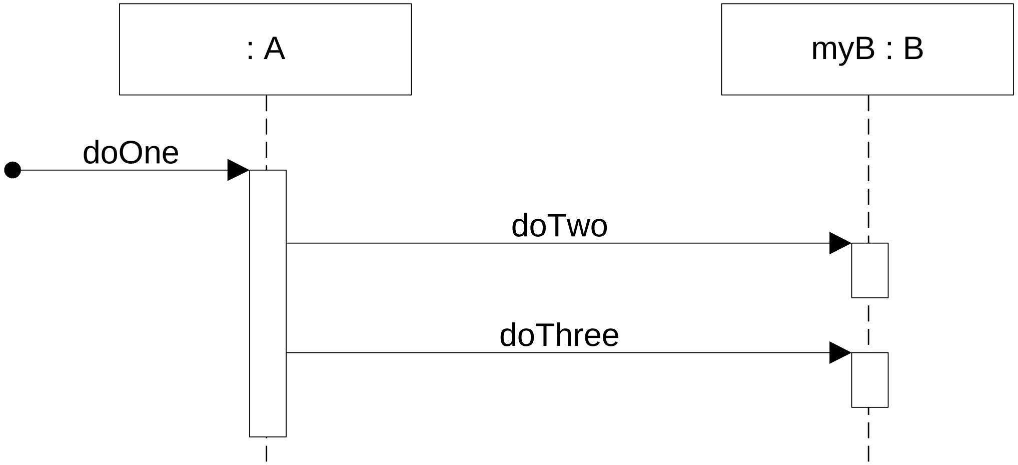
- Sequence diagram
- Communication diagram

- Interaction overview diagram
  - How a set of interaction diagrams are related in terms of logic / process flow

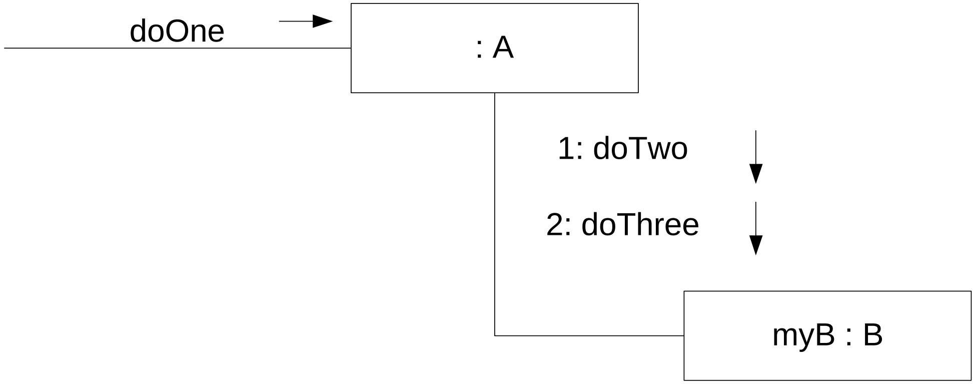
Interaction Diagram (Contd.)

- Sequence diagram
  - Illustrate object interactions in a kind of fence format
  - New objects are added to the right
  - Excellent for documentation
- Communication diagram
  - Illustrate object interactions in a graph / network format
  - Objects can be placed anywhere
  - Space-efficient

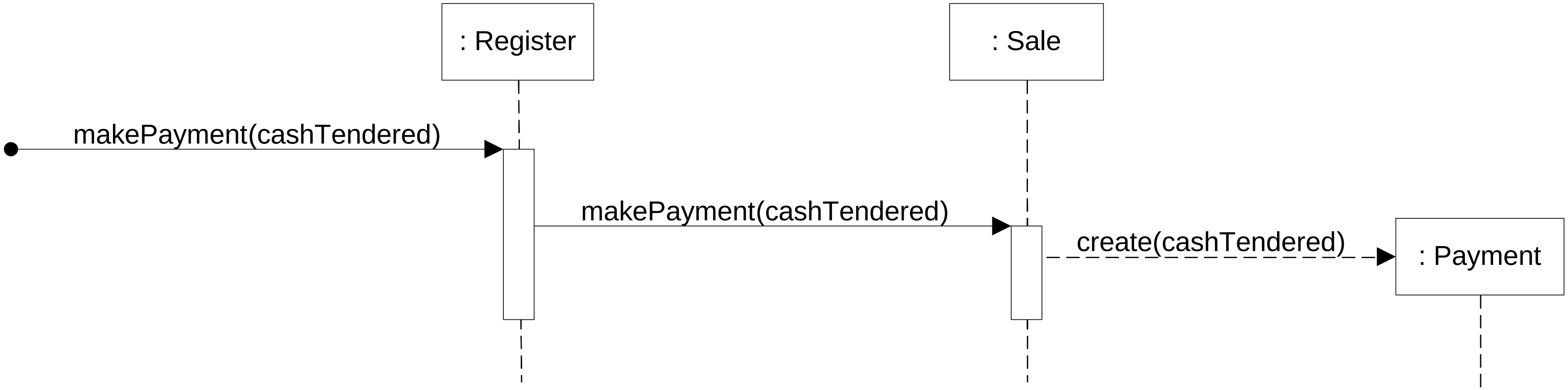
Interaction diagrams and Code fragment



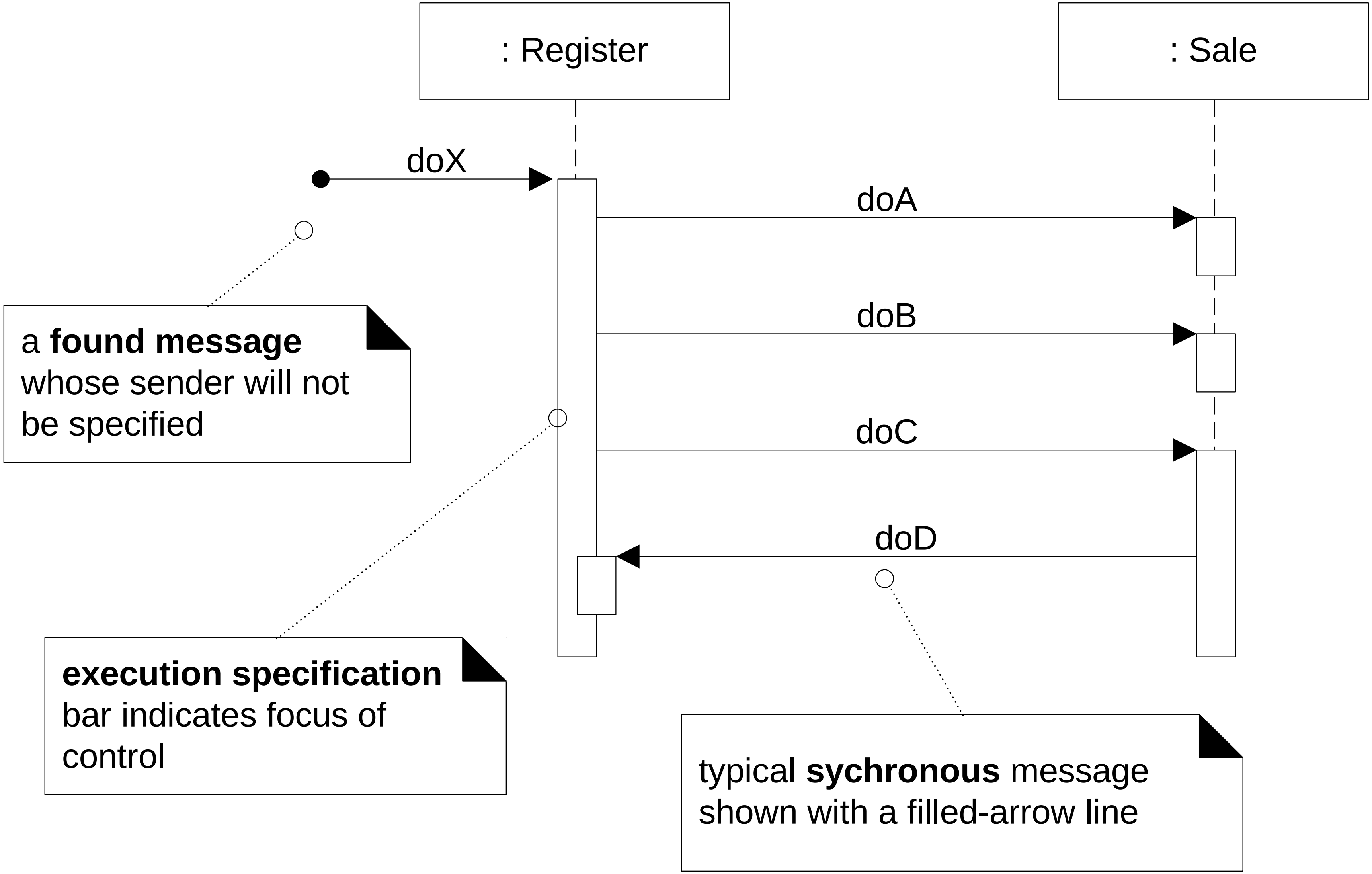
```
Public class A{
        private B myB = new B();
        public void done(){
                myB.doTwo();
                myB.doThree();
        }
}
```



Code fragment?



Sequence diagram: Execution specification

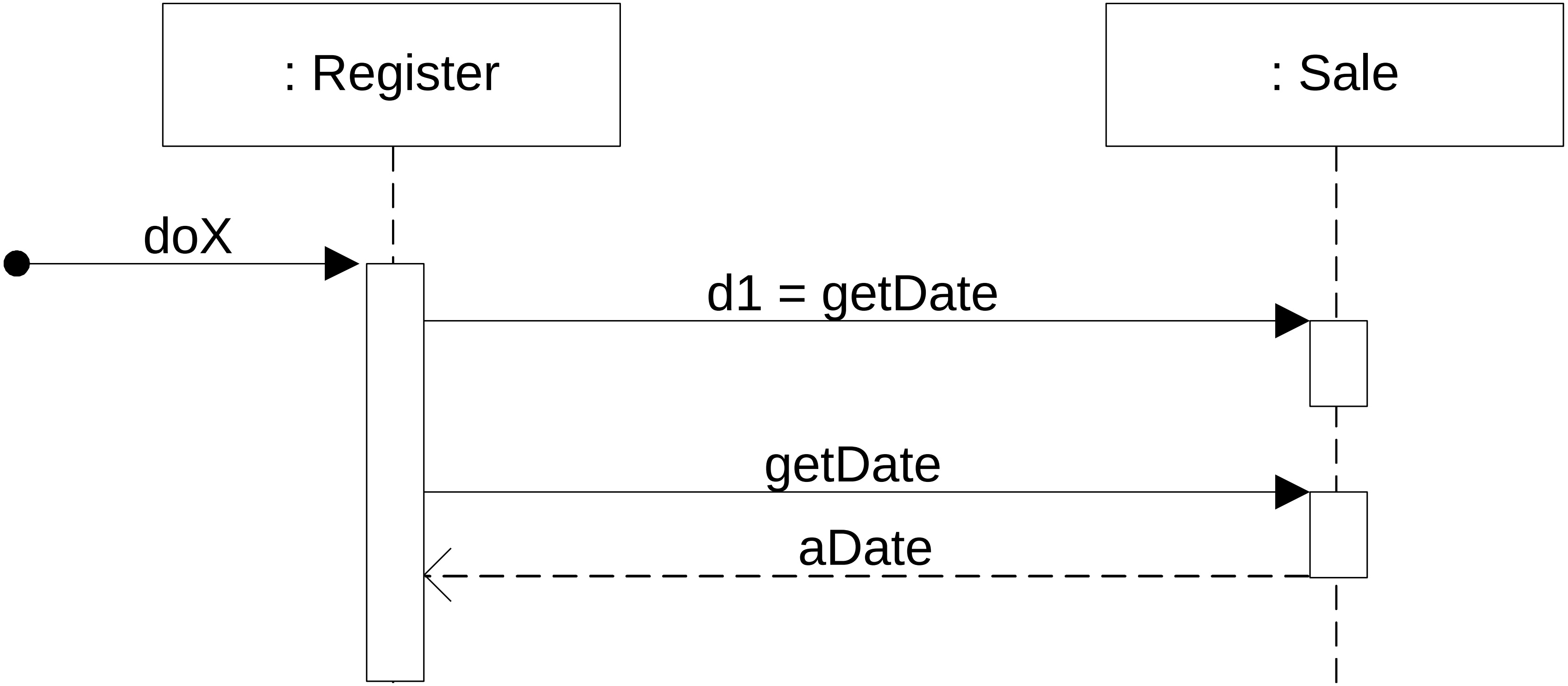


a **found message** whose sender will not be specified

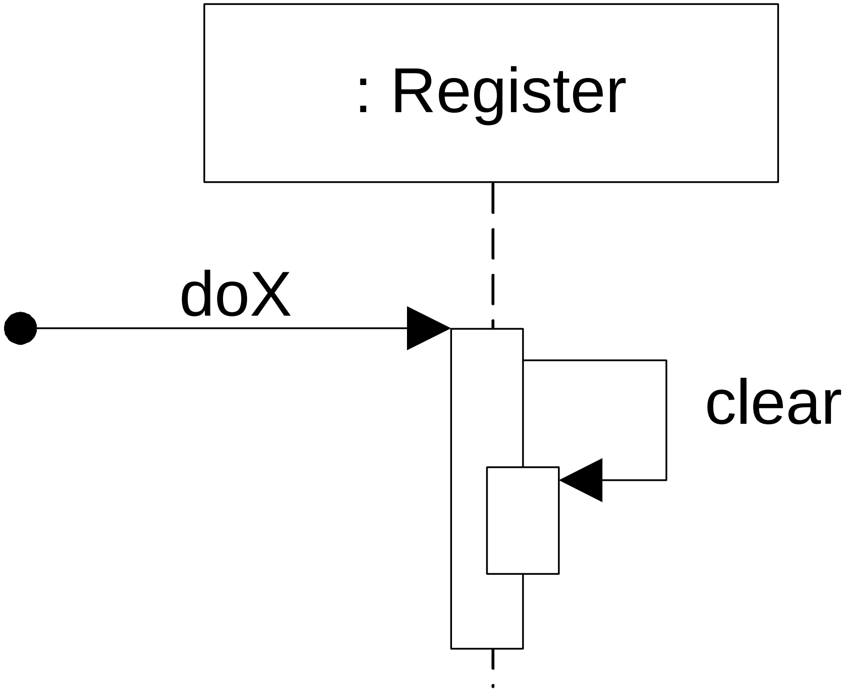
execution specification bar indicates focus of control

typical **sychronous** message shown with a filled-arrow line

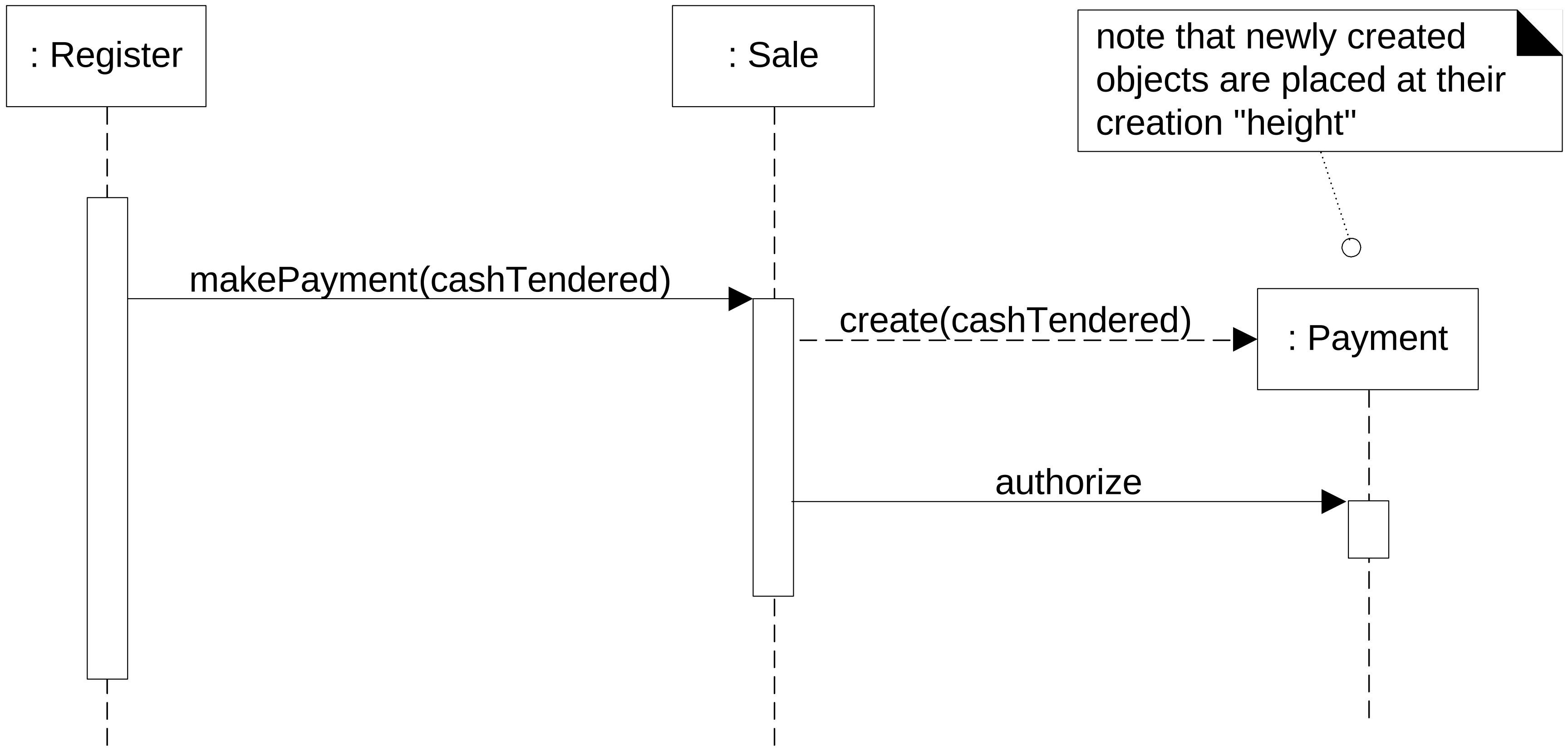
Sequence diagram: Reply / Return



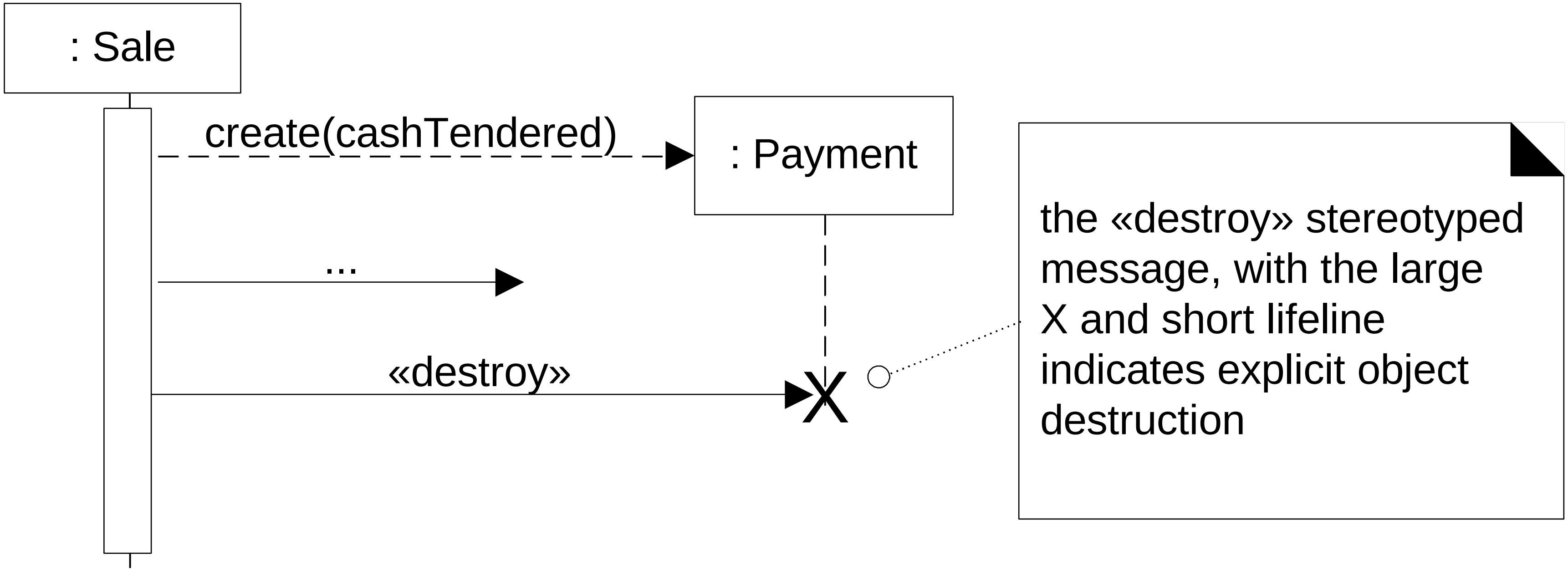
Sequence diagram: Messages to self / this



Sequence diagram: Creation of instances


: Register
: Sale
note that newly created objects are placed at their creation "height"
makePayment(cashTendered)
create(cashTendered)
: Payment
authorize

Sequence diagram: Destruction of instances

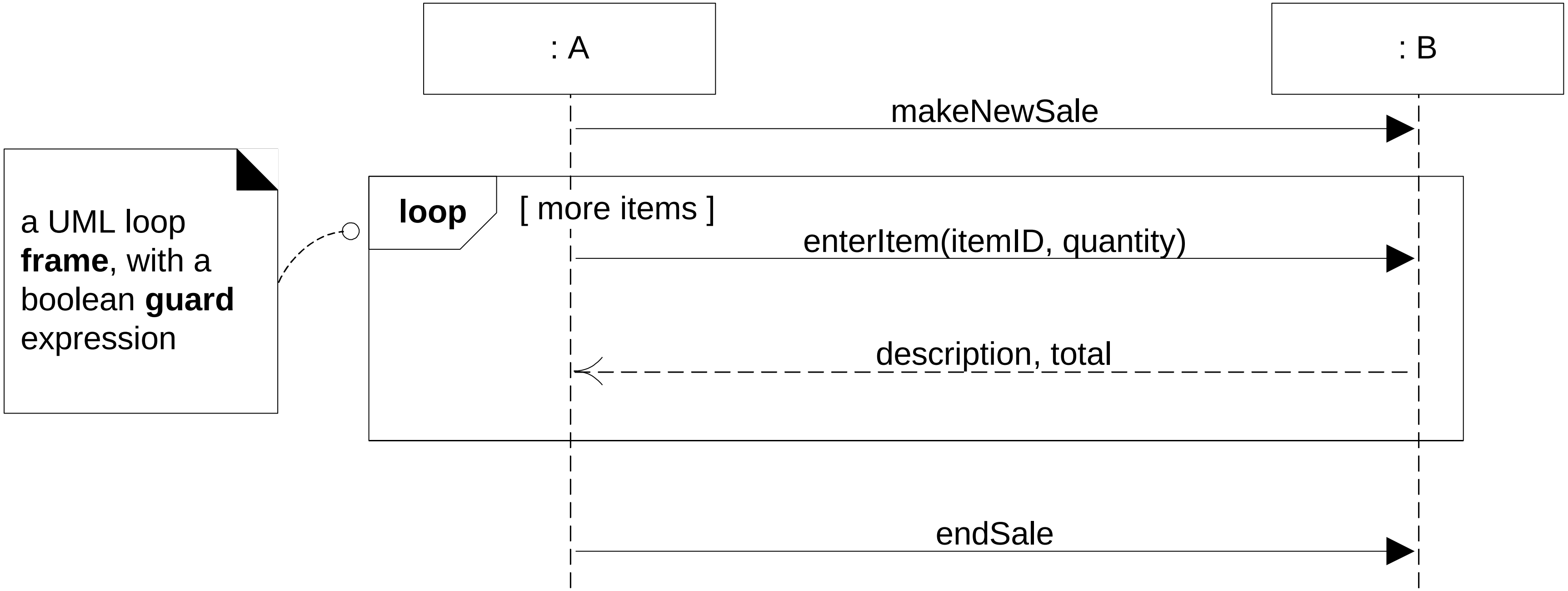


the «destroy» stereotyped message, with the large X and short lifeline indicates explicit object destruction

Sequence diagram: Interaction frames

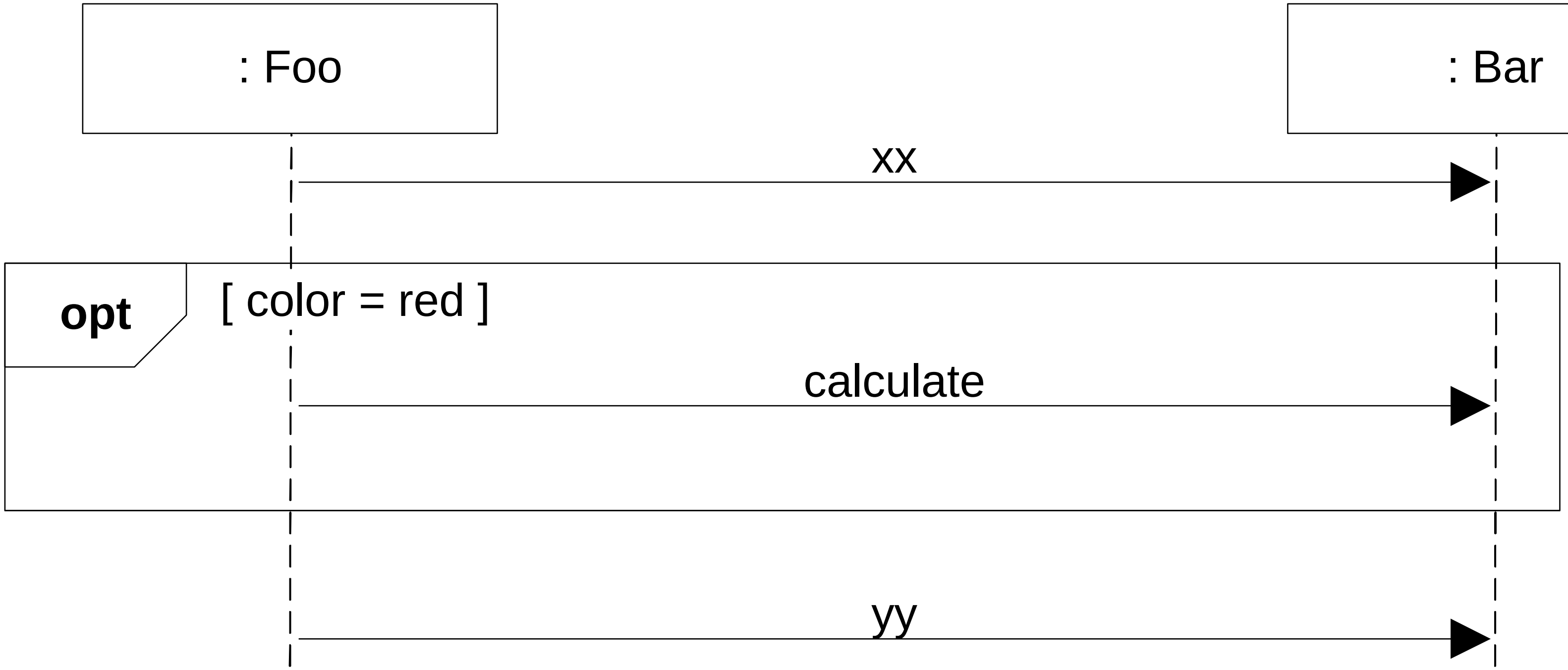
- Alt – If/Else
- Loop – Loop
- Opt – If
- Par - Concurrent
- Region – Critical section

Sequence diagram: Loop

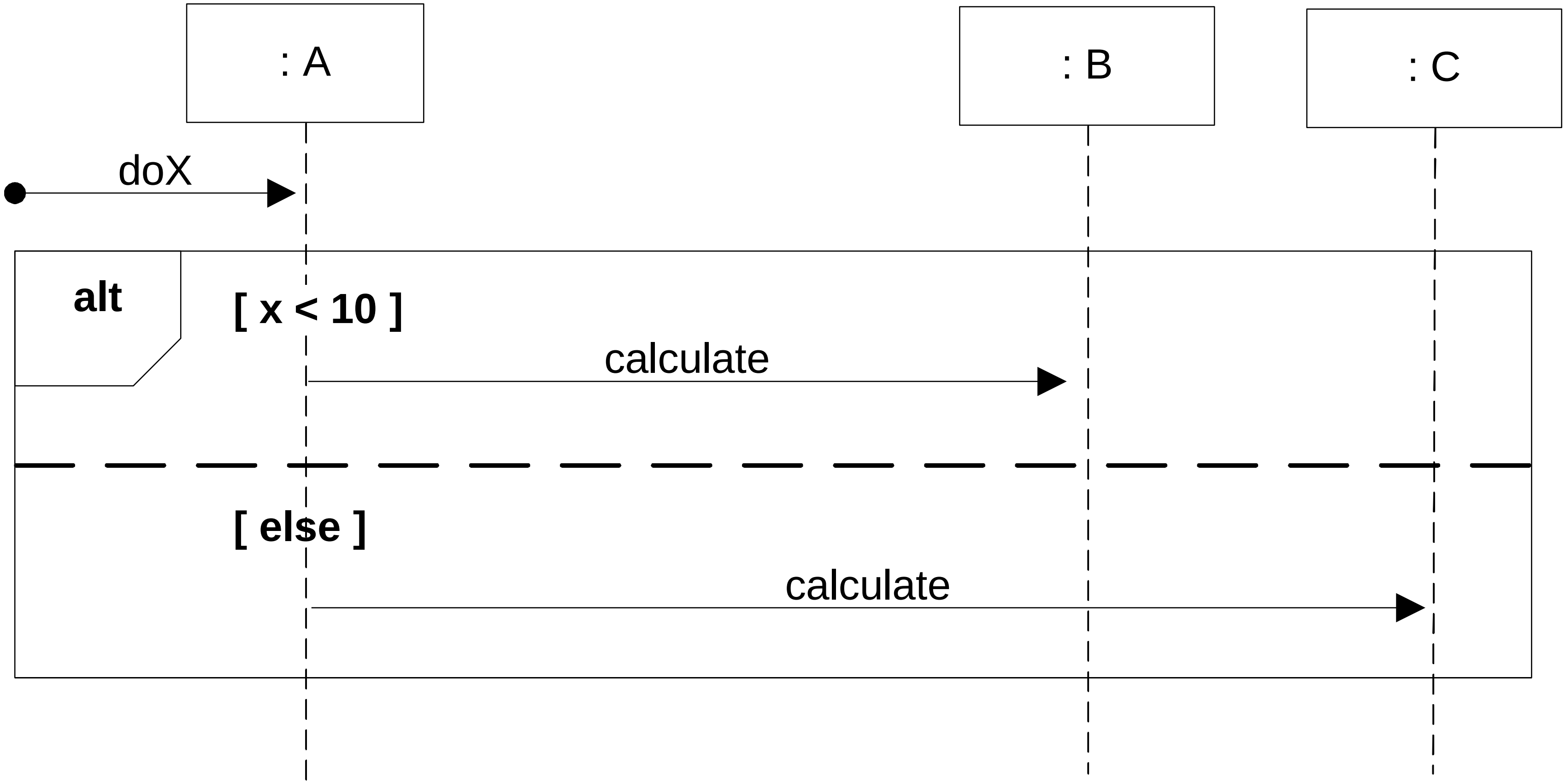


a UML loop **frame**, with a boolean **guard** expression

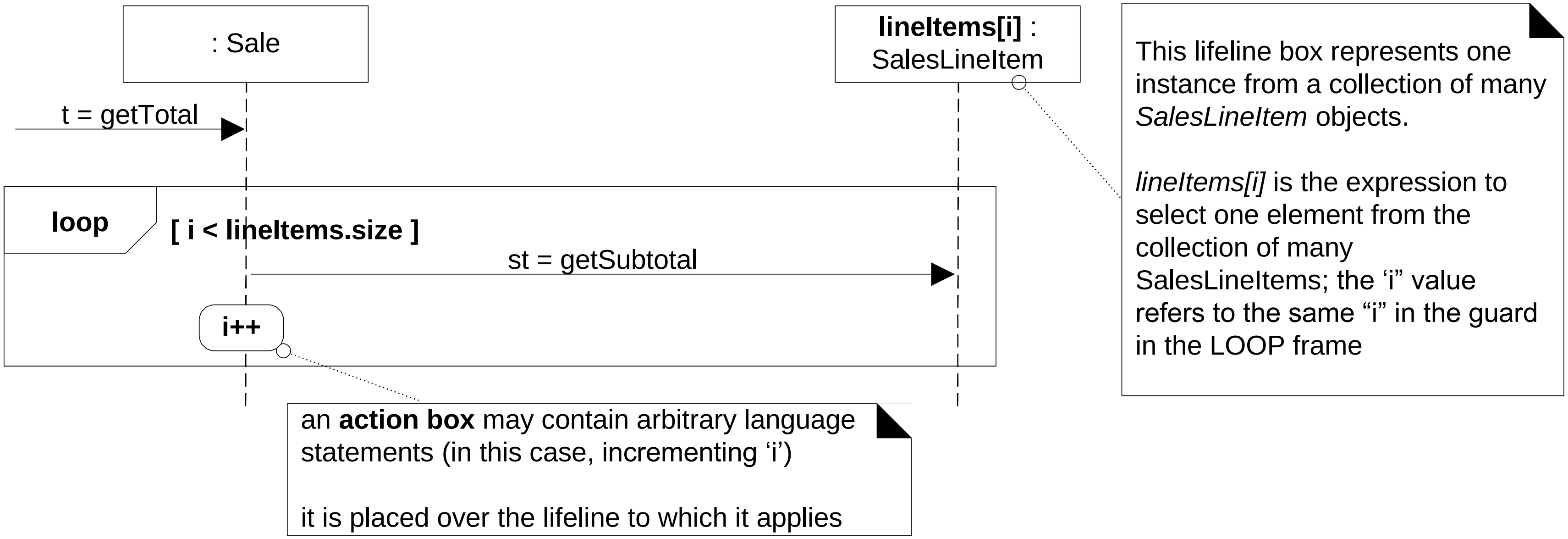
Sequence diagram: Opt



Sequence diagram: Alt



Sequence diagram: Iteration over a collection



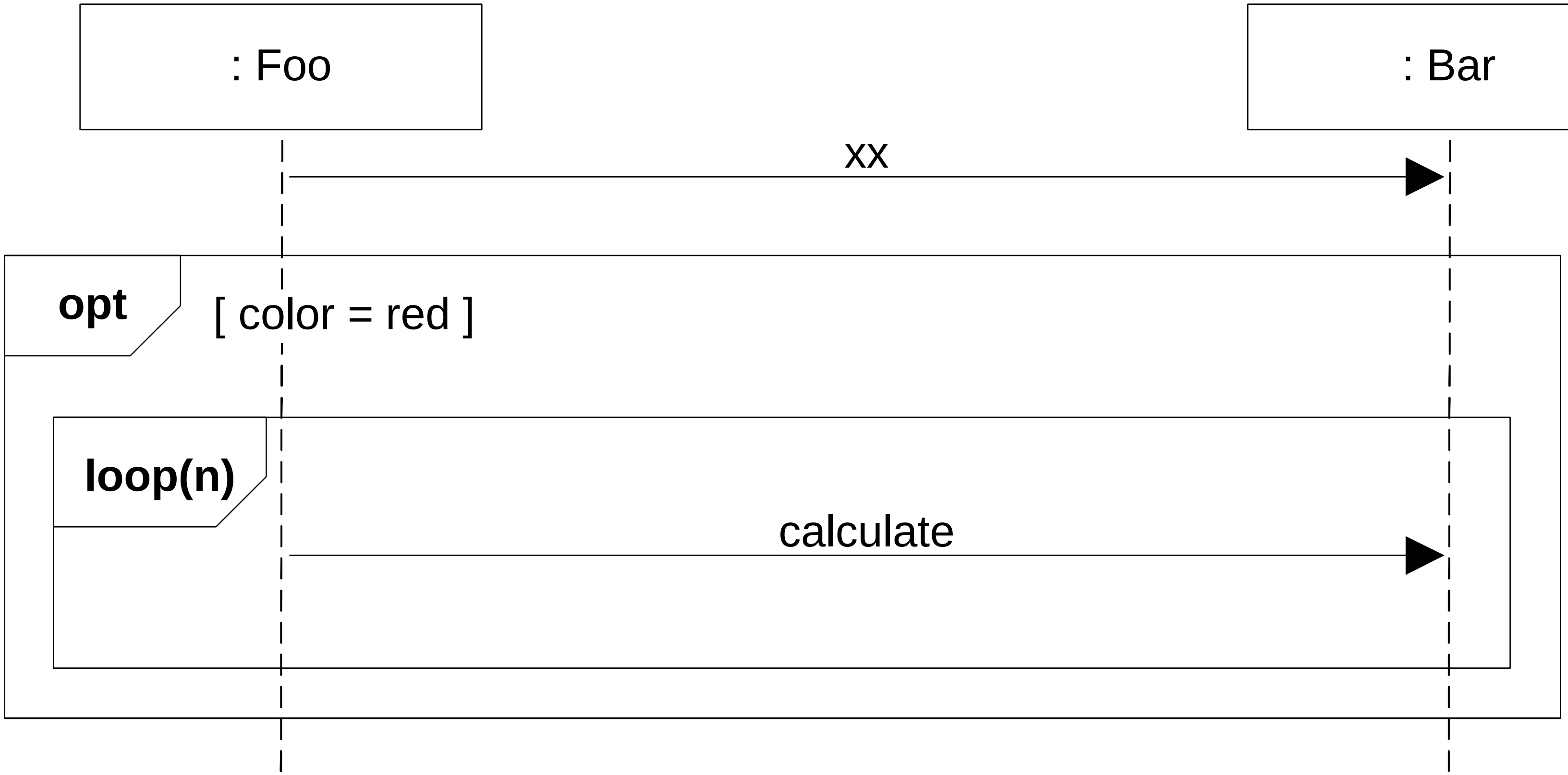
This lifeline box represents one instance from a collection of many *SalesLineItem* objects.

lineItems[i] is the expression to select one element from the collection of many SalesLineItems; the ‘i” value refers to the same “i” in the guard in the LOOP frame

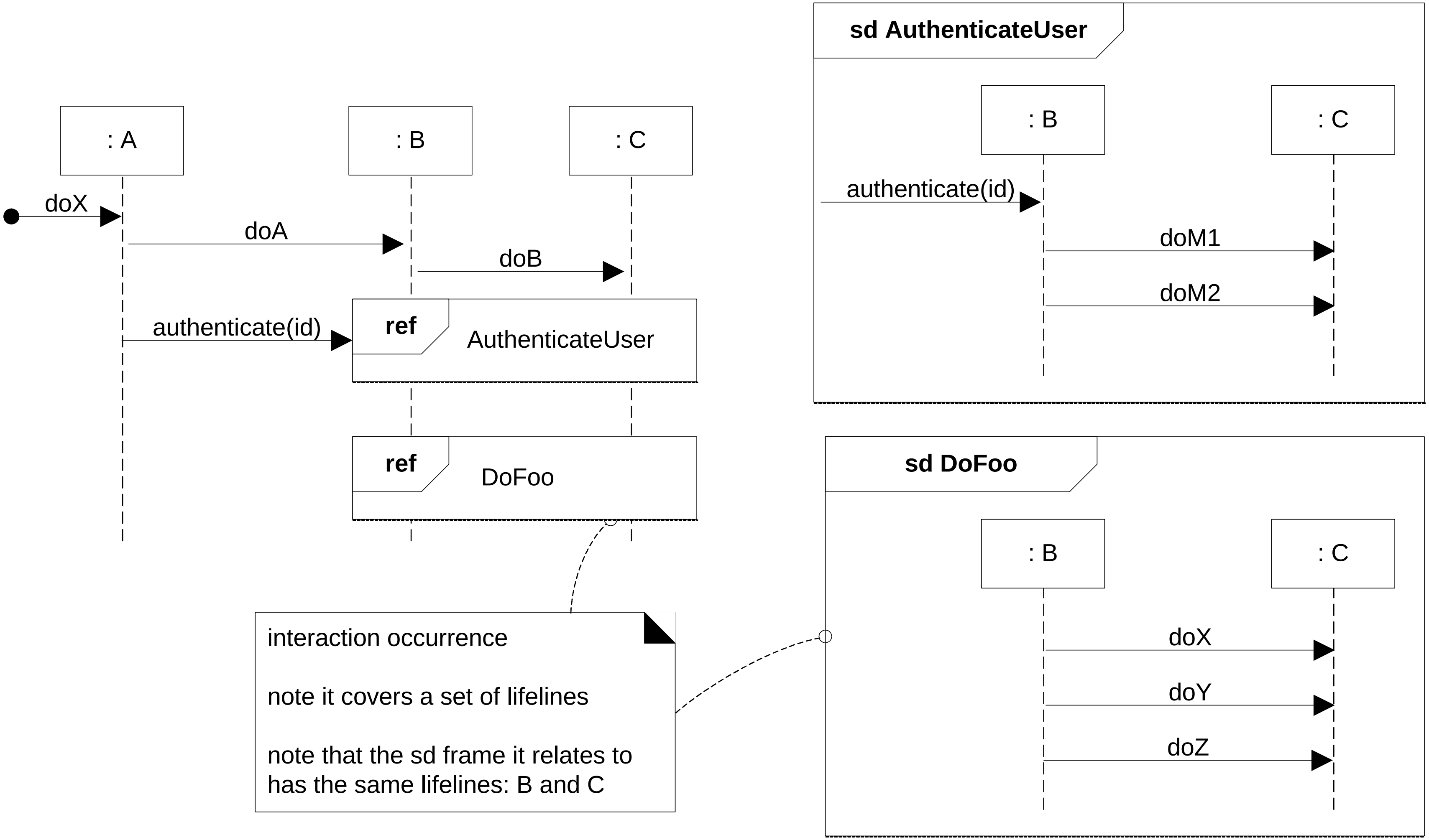
an **action box** may contain arbitrary language statements (in this case, incrementing ‘i’)

it is placed over the lifeline to which it applies

Sequence diagram: Nesting over frames



Sequence diagram: Related frames

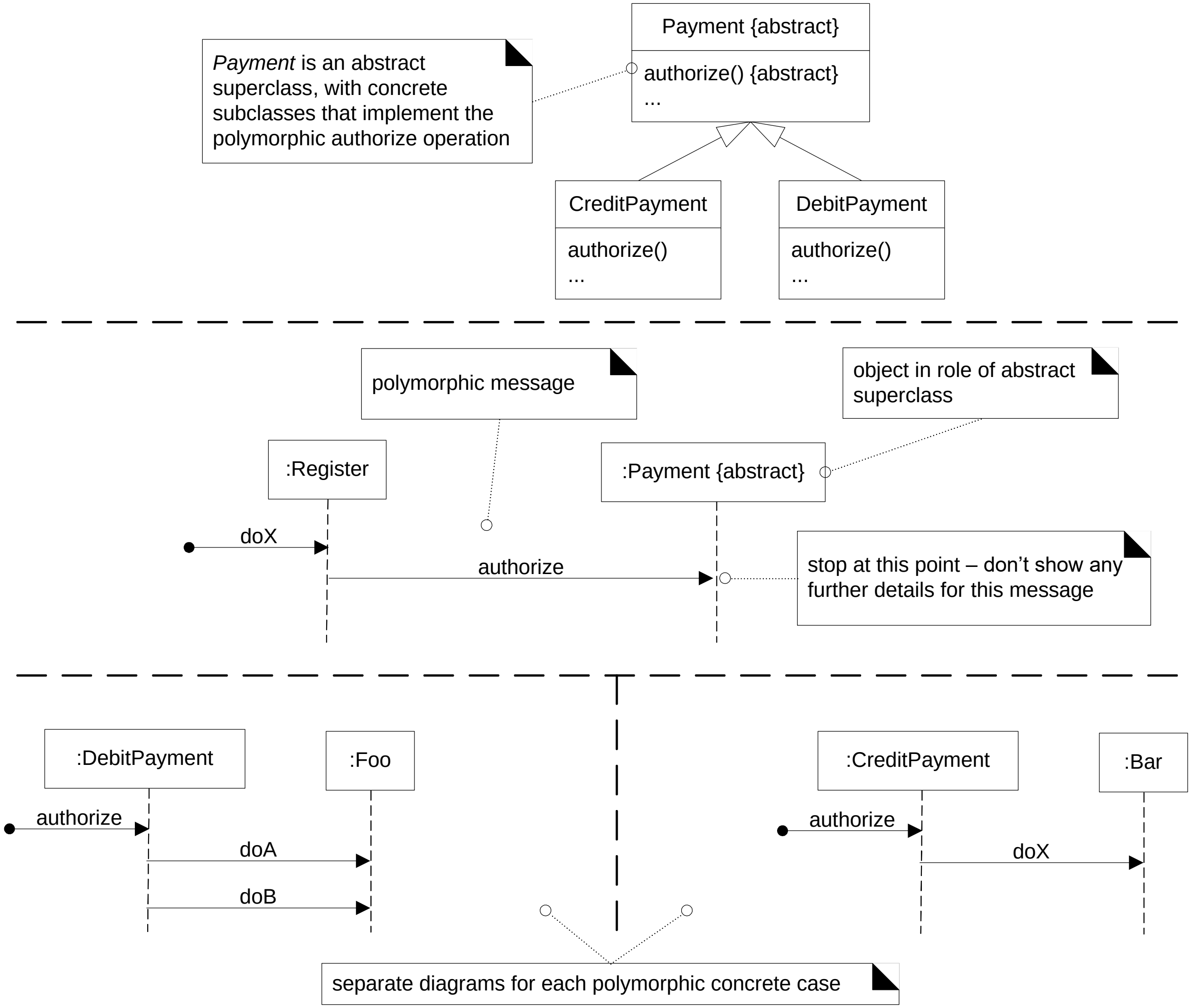


interaction occurrence

note it covers a set of lifelines

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

Sequence diagram: Polymorphic calls



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

polymorphic message

object in role of abstract superclass

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

separate diagrams for each polymorphic concrete case

Sequence diagram: Synchronous / Asynchronous calls

a stick arrow in UML implies an asynchronous call

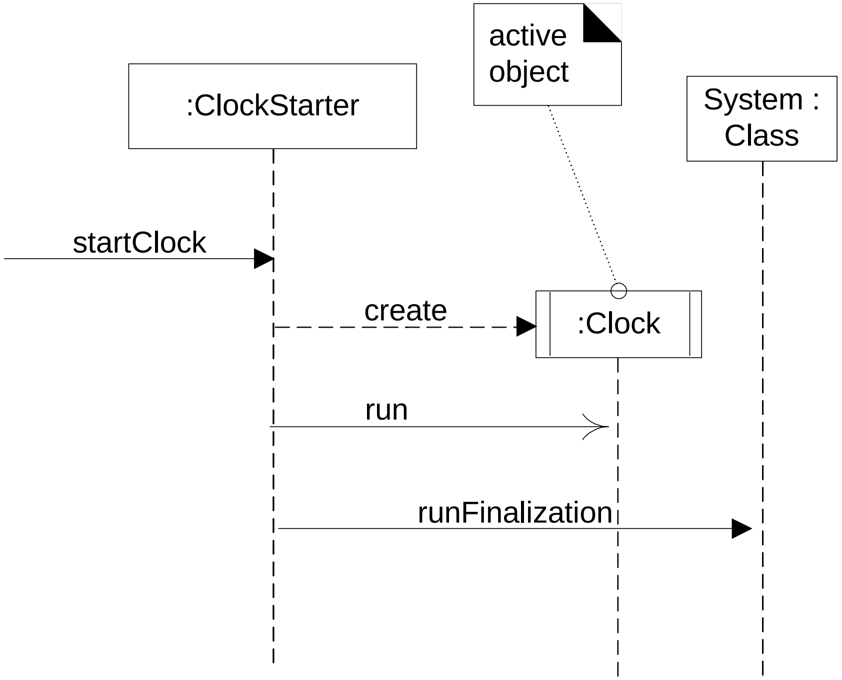
a filled arrow is the more common synchronous call

In Java, for example, an asynchronous call may occur as follows:

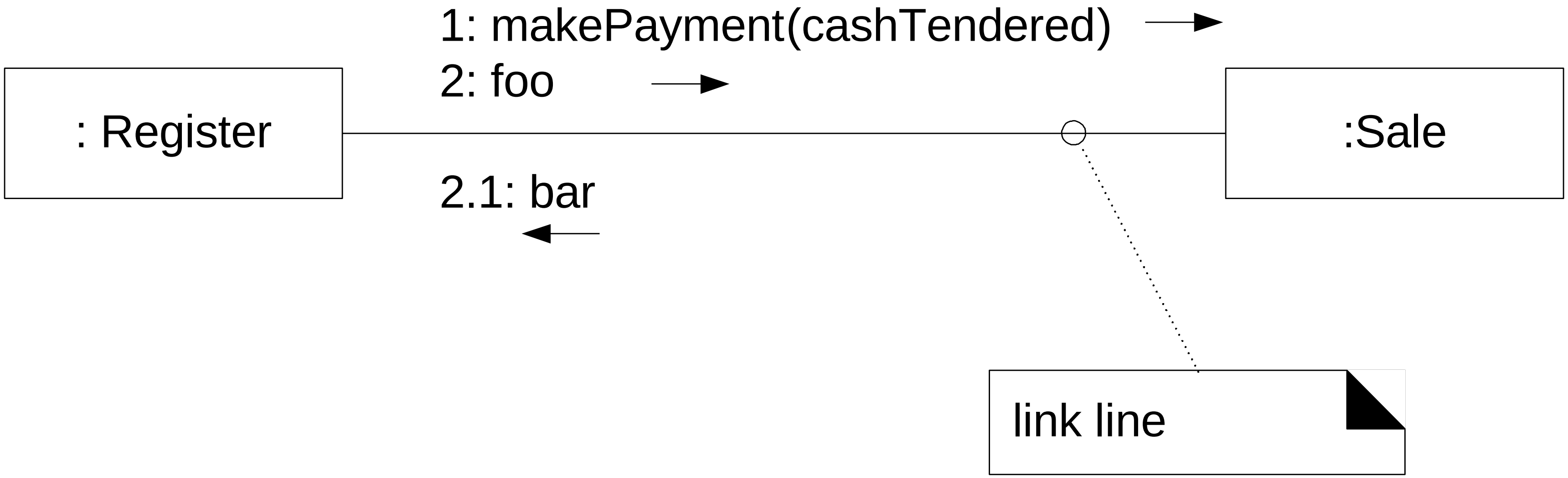
```
// Clock implements the Runnable interface
Thread t = new Thread( new Clock() );
t.start();
```

the asynchronous *start* call always invokes the *run* method on the *Runnable* (*Clock*) object

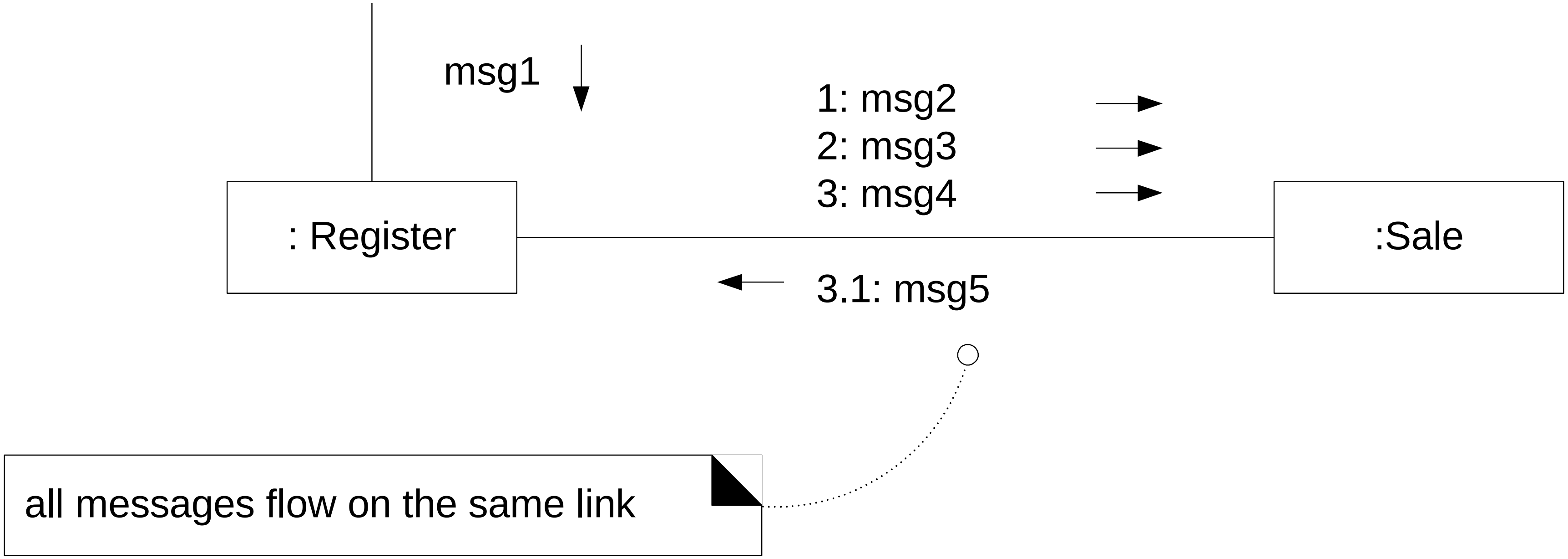
to simplify the UML diagram, the *Thread* object and the *start* message may be avoided (they are standard “overhead”); instead, the essential detail of the *Clock* creation and the *run* message imply the asynchronous call



Communication diagram: Link lines

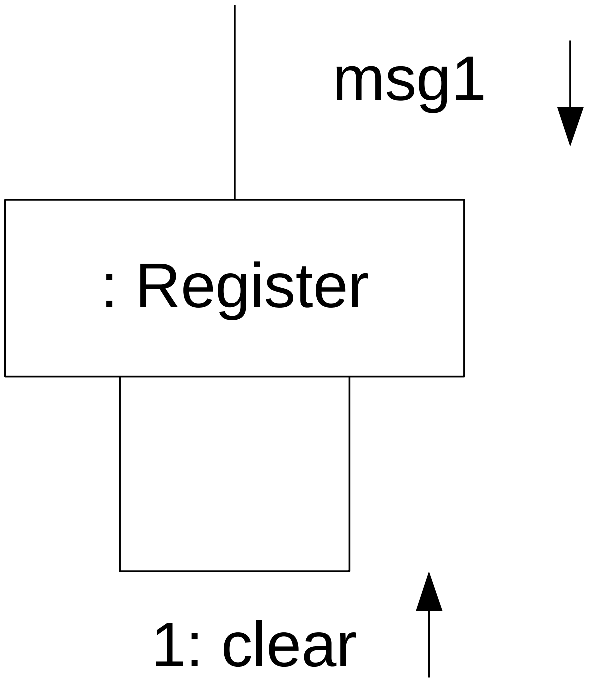


Communication diagram: Messages

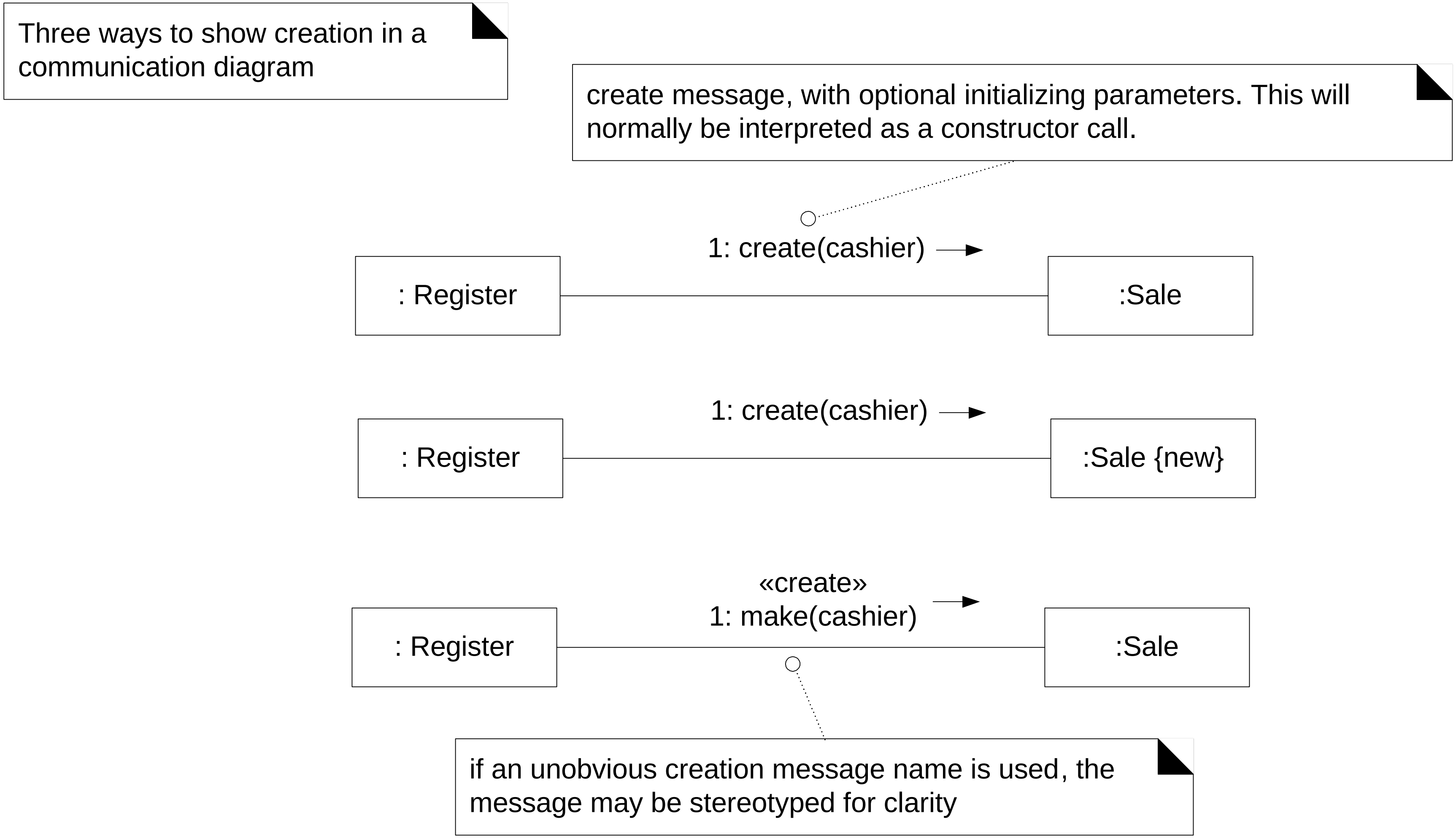


all messages flow on the same link

Communication diagram: Messages to self / this



Communication diagram: Creation of instances



Three ways to show creation in a communication diagram

create message, with optional initializing parameters. This will normally be interpreted as a constructor call.

1: create(cashier) →

: Register

:Sale

1: create(cashier) →

: Register

:Sale {new}

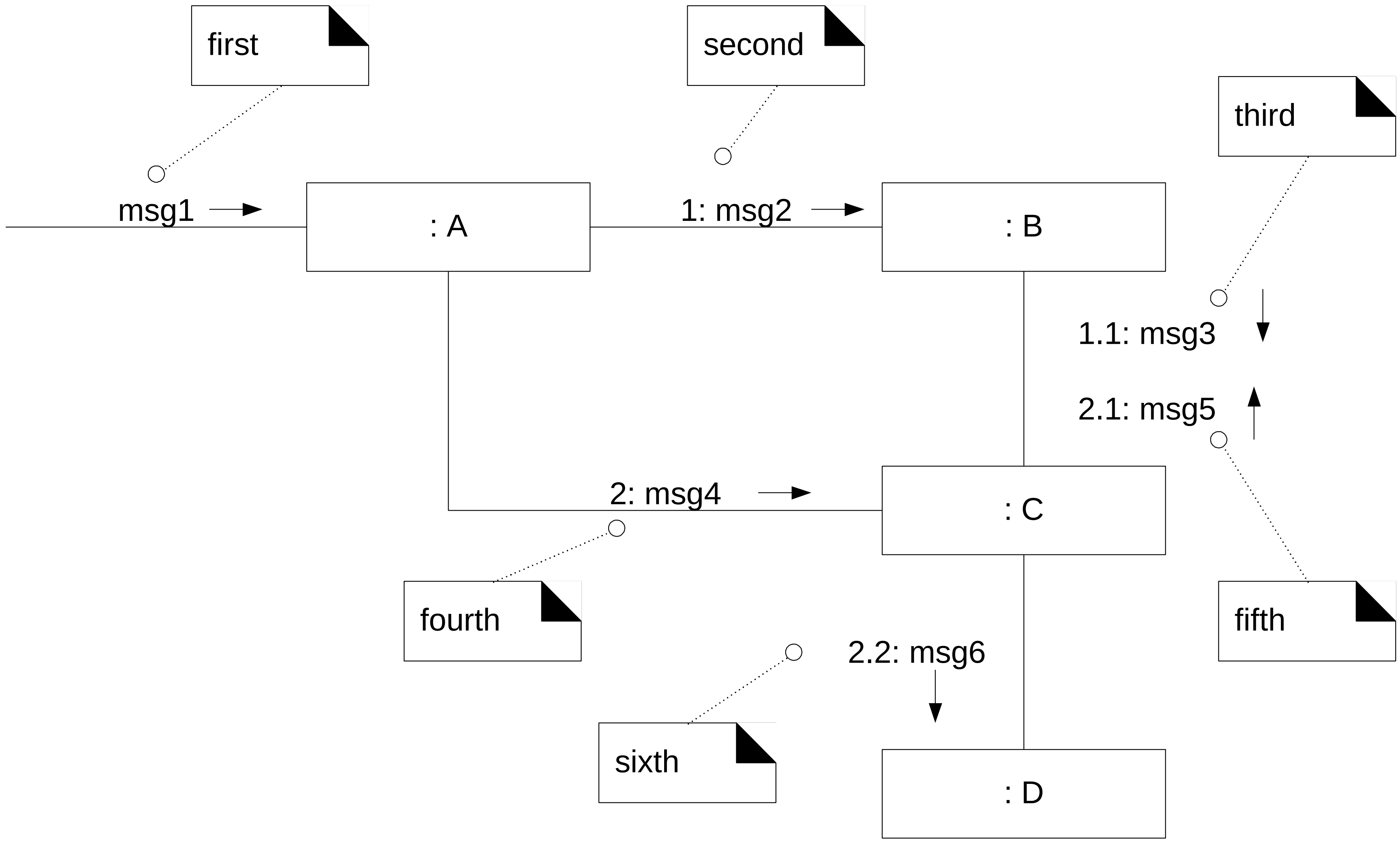
«create»
1: make(cashier) →

: Register

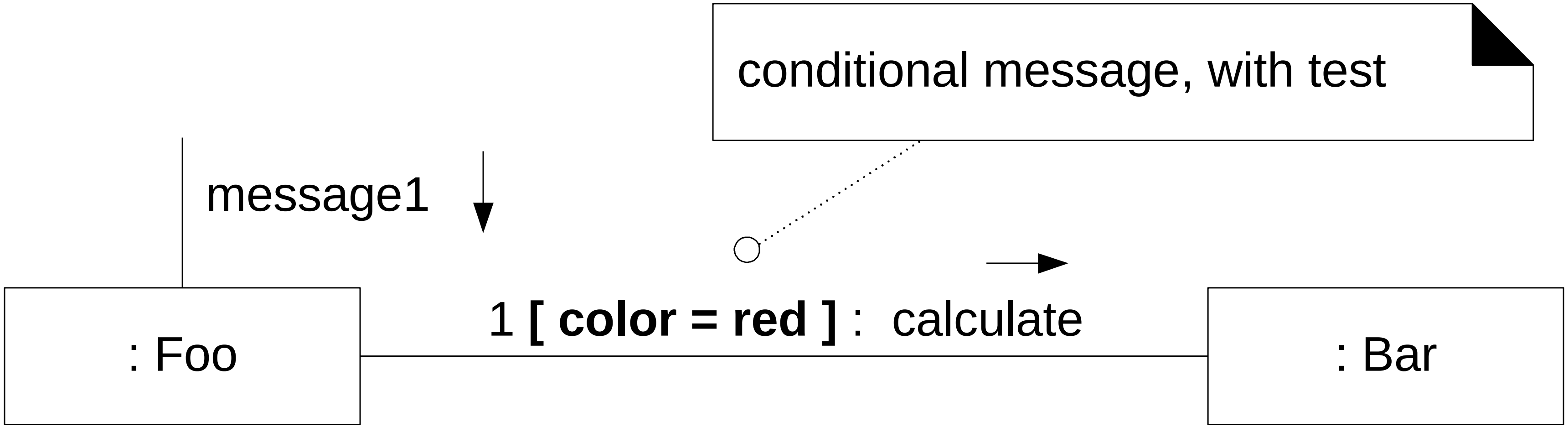
:Sale

if an unobvious creation message name is used, the message may be stereotyped for clarity

Communication diagram: Message number sequencing



Communication diagram: Conditional message



conditional message, with test

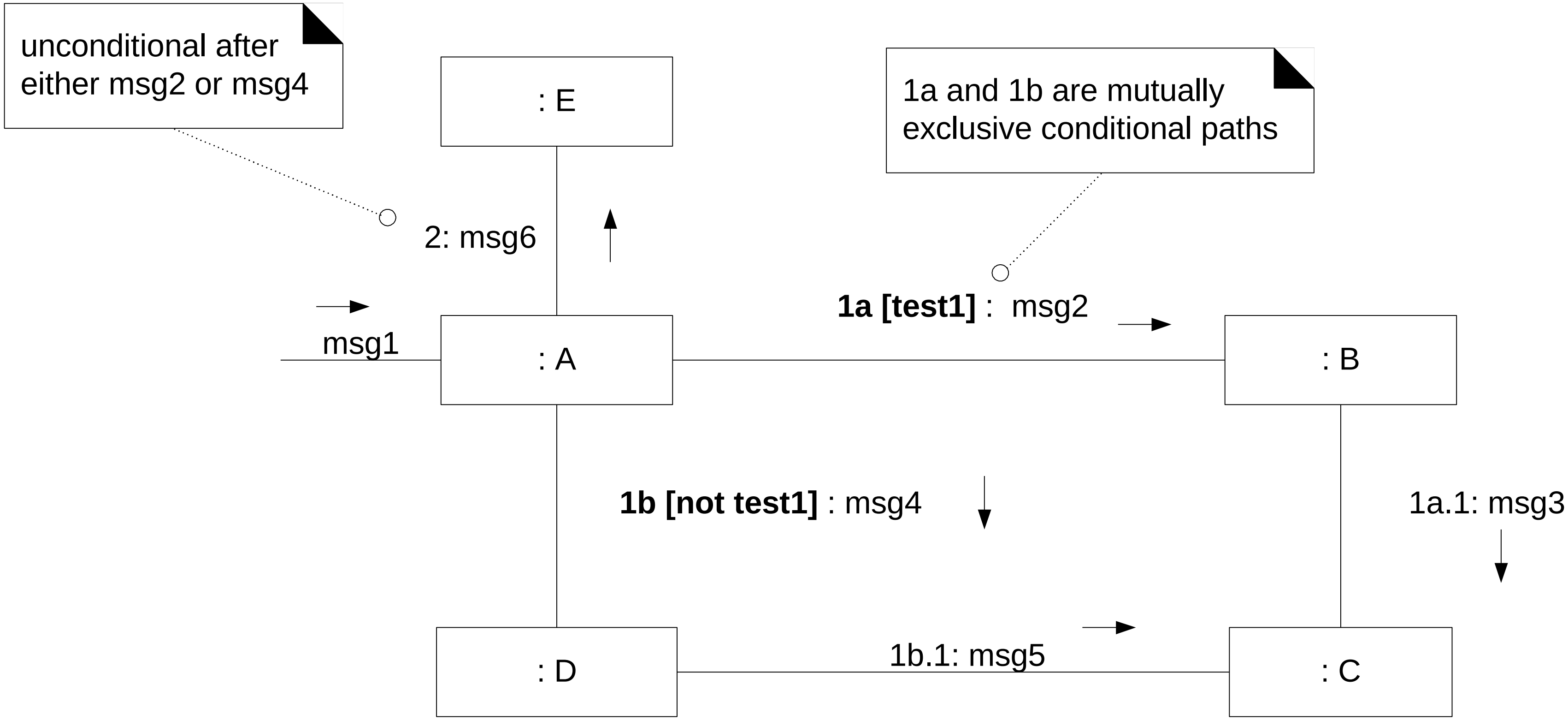
message1

1 **[color = red]** : calculate

: Foo

: Bar

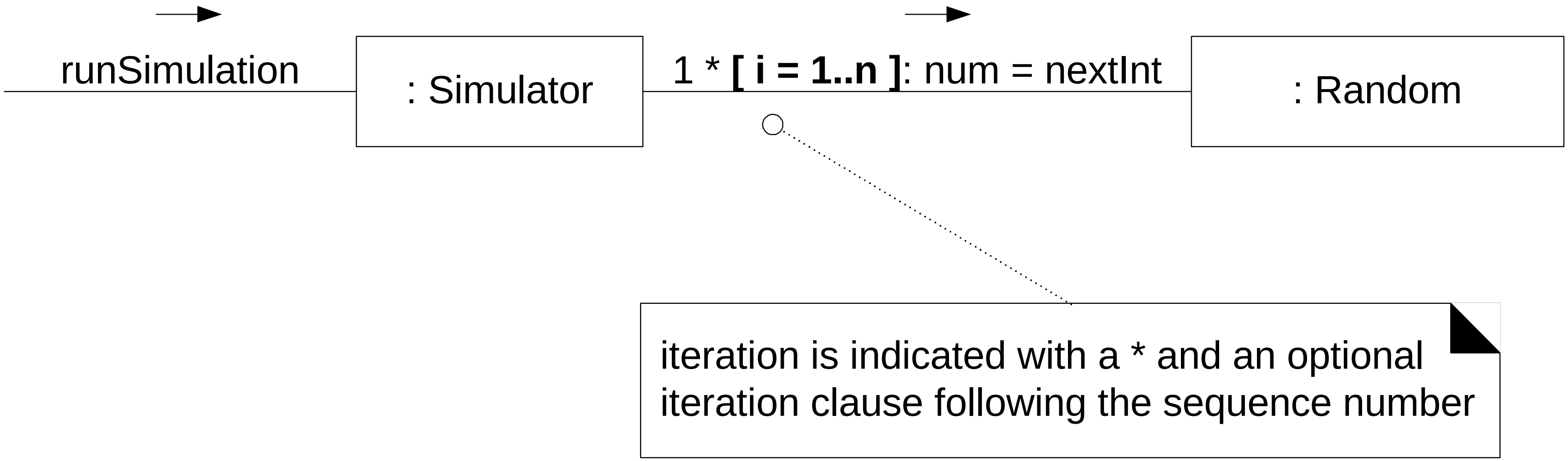
Communication diagram: Mutually exclusive messages



unconditional after either msg2 or msg4

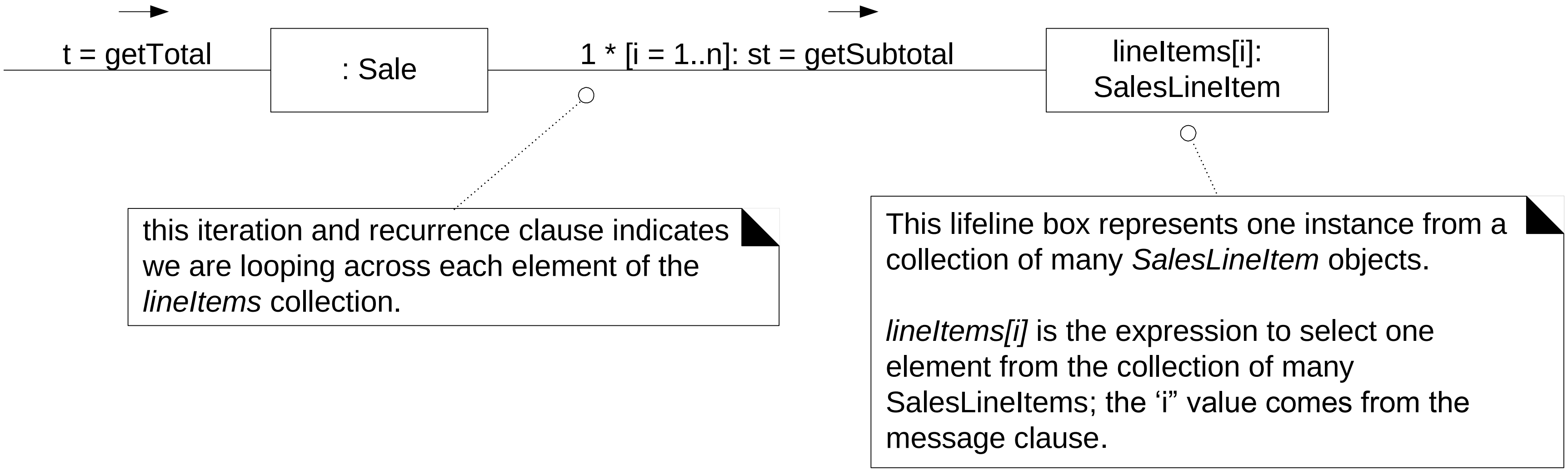
1a and 1b are mutually exclusive conditional paths

Communication diagram: Iteration / Loop



iteration is indicated with a * and an optional iteration clause following the sequence number

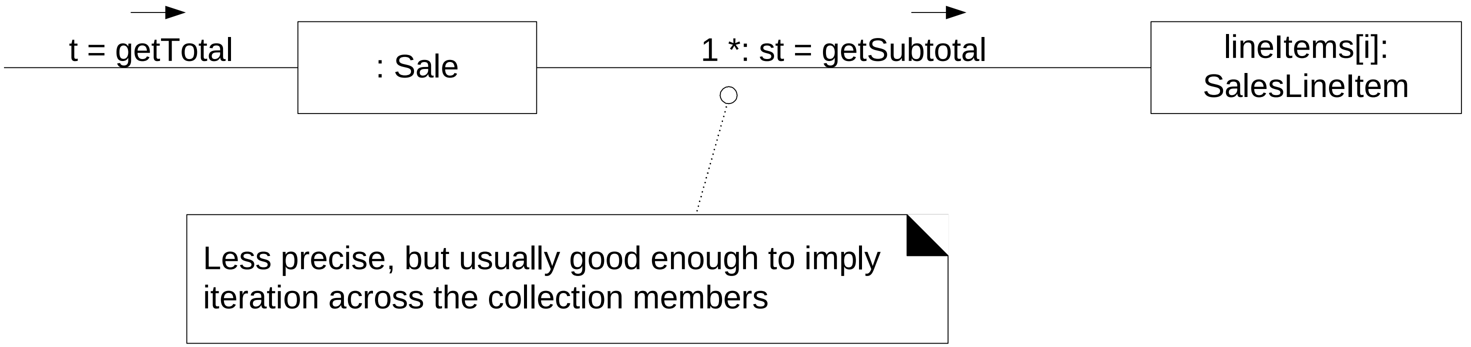
Communication diagram: Iteration over a collection



this iteration and recurrence clause indicates we are looping across each element of the *lineItems* collection.

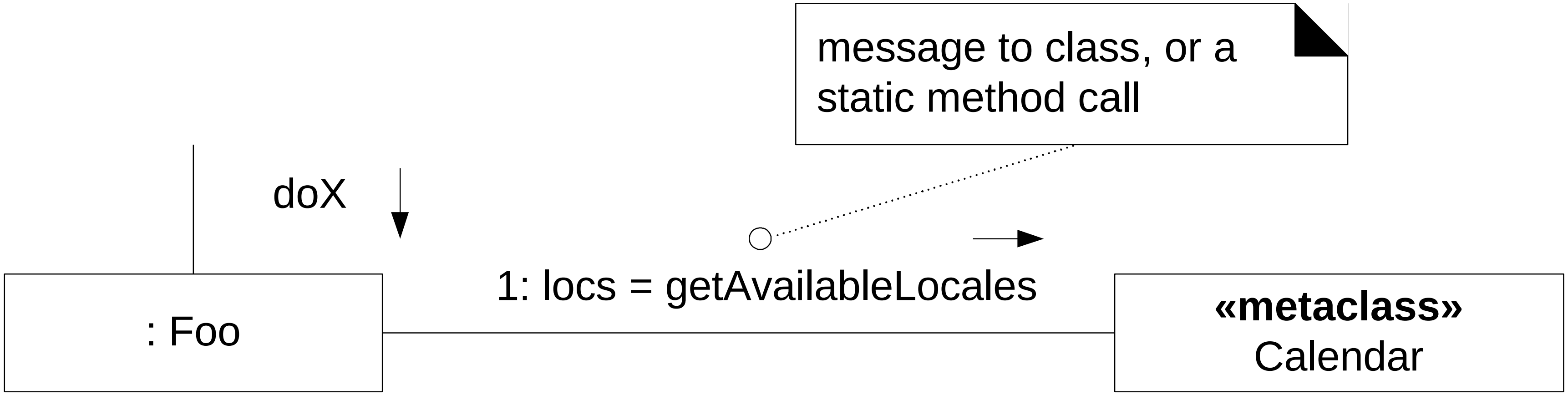
This lifeline box represents one instance from a collection of many *SalesLineItem* objects.

lineItems[i] is the expression to select one element from the collection of many SalesLineItems; the ‘i” value comes from the message clause.



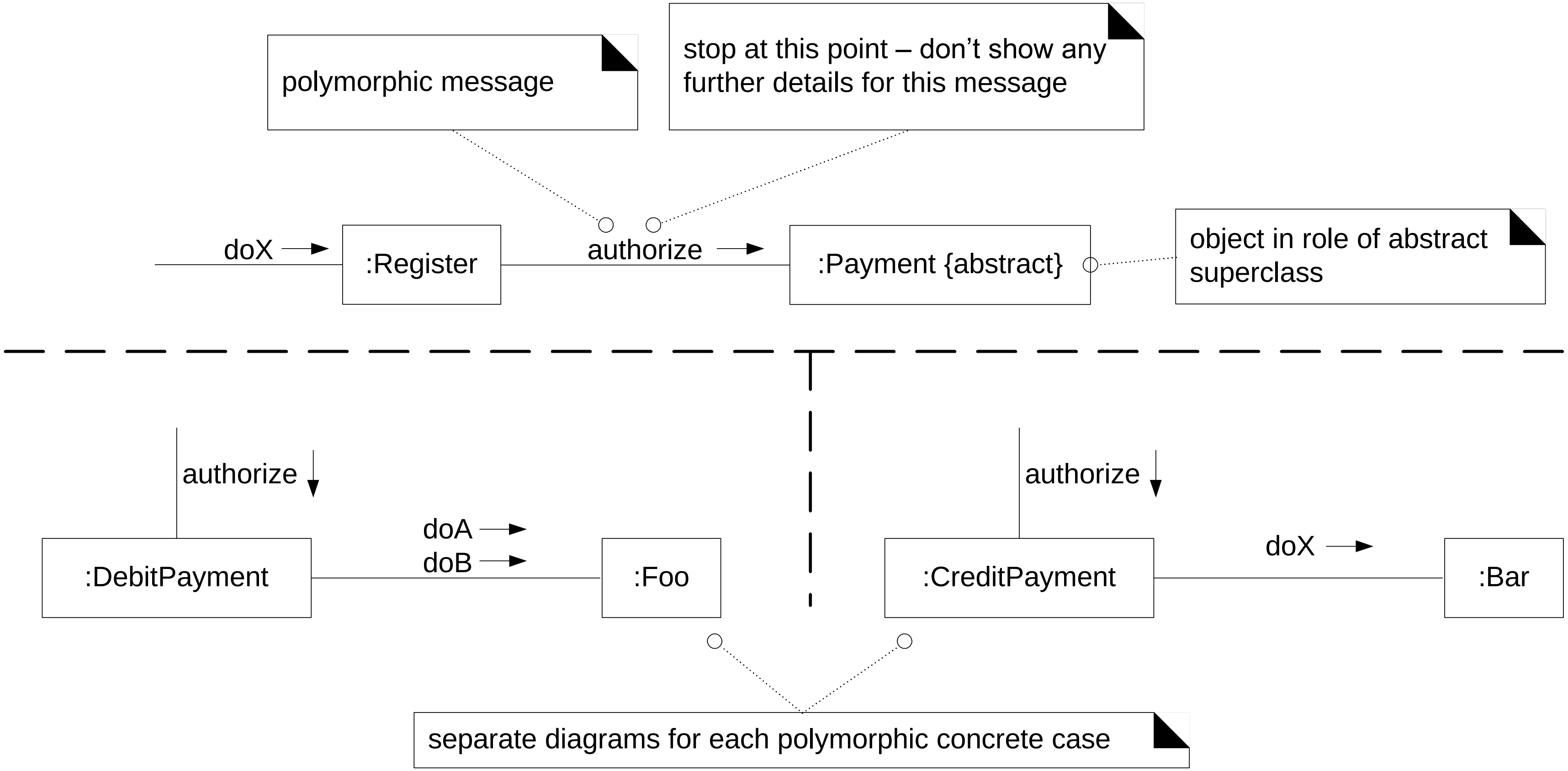
Less precise, but usually good enough to imply iteration across the collection members

Communication diagram: Static object



message to class, or a static method call

Communication diagram: Polymorphic calls



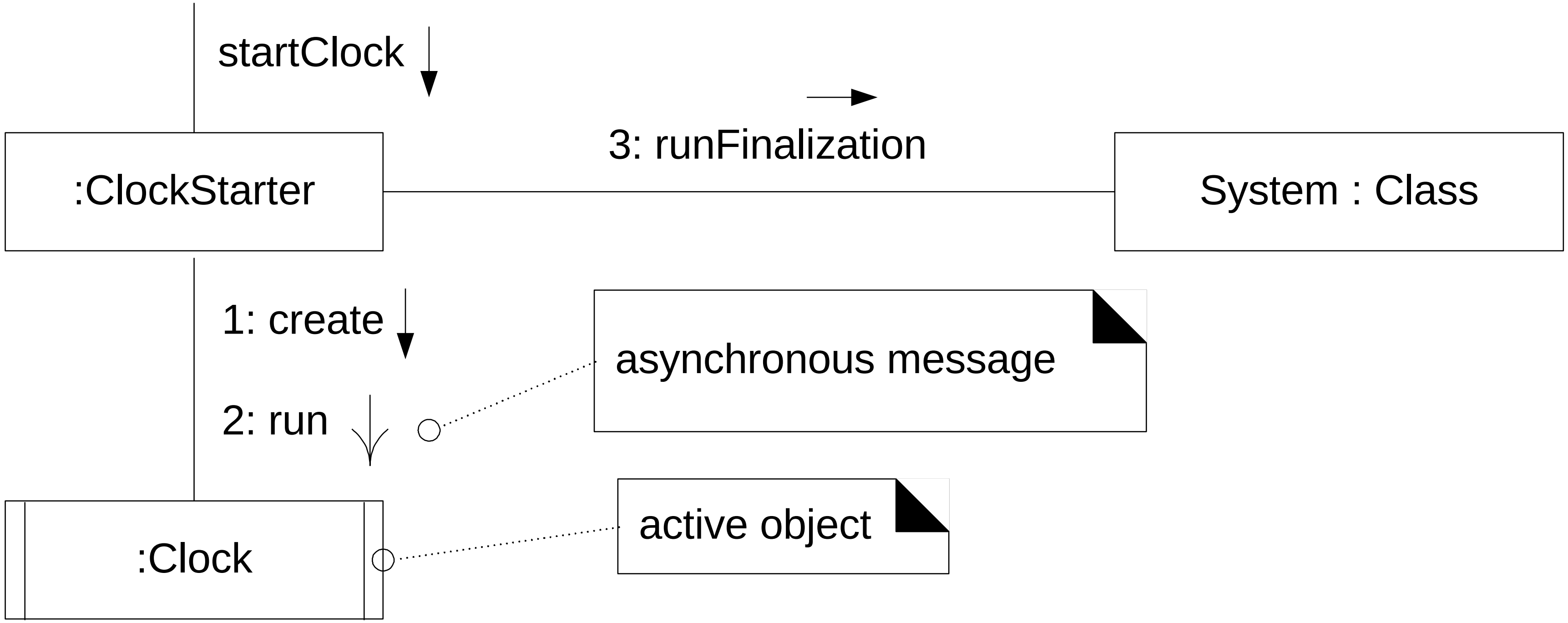
polymorphic message

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

object in role of abstract superclass

separate diagrams for each polymorphic concrete case

Communication diagram: Synchronous / Asynchronous calls



Next sessions…

- Interaction and Class Design: Introduction

Reading assignment

- Reference Book
  - Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and the Unified Process, Second Edition, Craig Larman, 2004
    - Chapter 6: Use Cases: Pages 66.
    - Chapter 10: System Sequence Diagrams: Pages 173-180.
    - Chapter 11: Operation Contracts: Pages 181-194.
    - Chapter 13: Logical Architecture and UML Package Diagrams: Pages 197-212.
    - Chapter 15: UML Interaction Diagrams: Pages 221-247.