Software Design and Analysis

Interaction Diagrams

Sequence Diagram & Communication Diagram

Dr. Syed Muazzam Ali Shah

Assistant Professor

Department of Computer Science

National University of Computer and Emerging Sciences

Interaction Diagrams

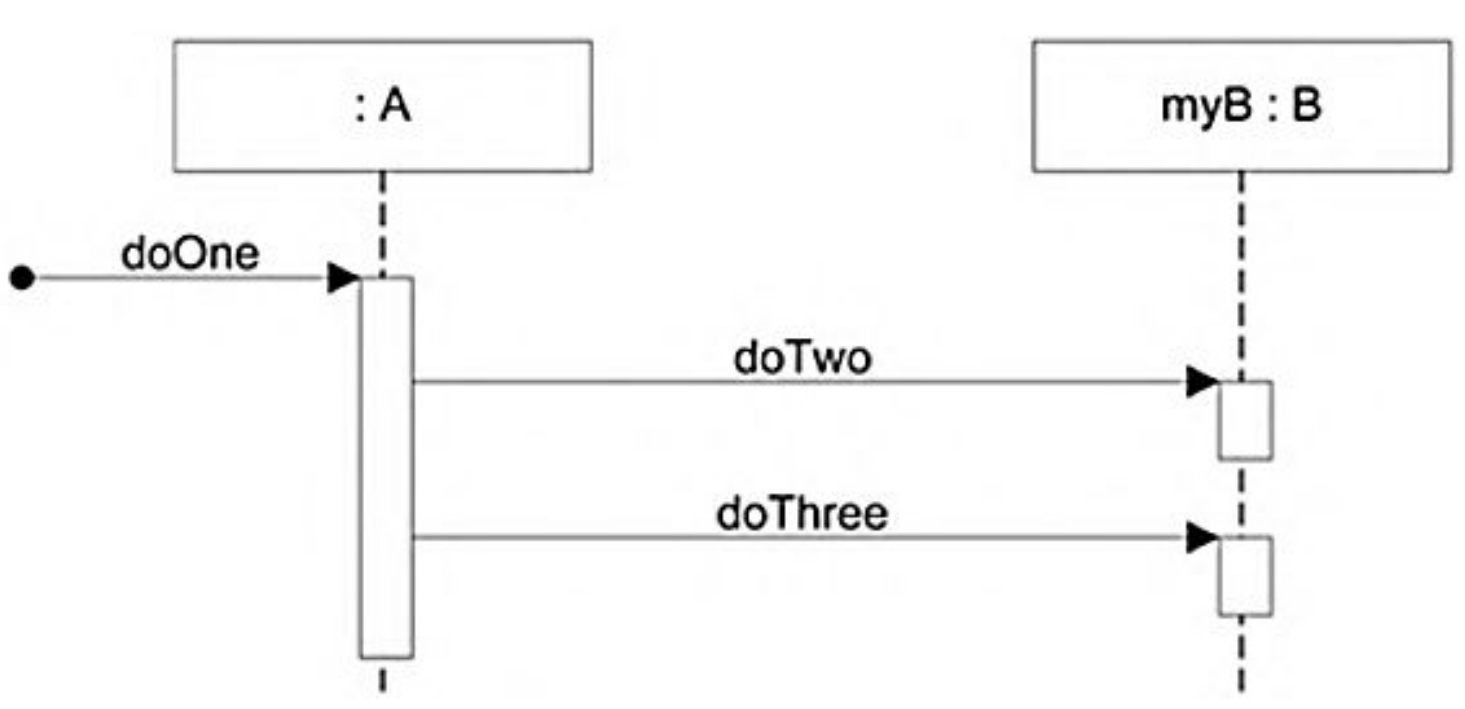
- The term interaction diagram, is a generalization of two more specialized UML diagram types:
  - Sequence diagrams
  - Communication/Collaboration diagrams.
- Both can be used to express similar message interactions.
- Sequence diagrams are the more notationally rich of the two types.

Sequence Diagram

Sequence diagrams illustrate interactions in a kind of fence format, in which each new object is added to the right.

Sequence Diagram

Sequence Diagram



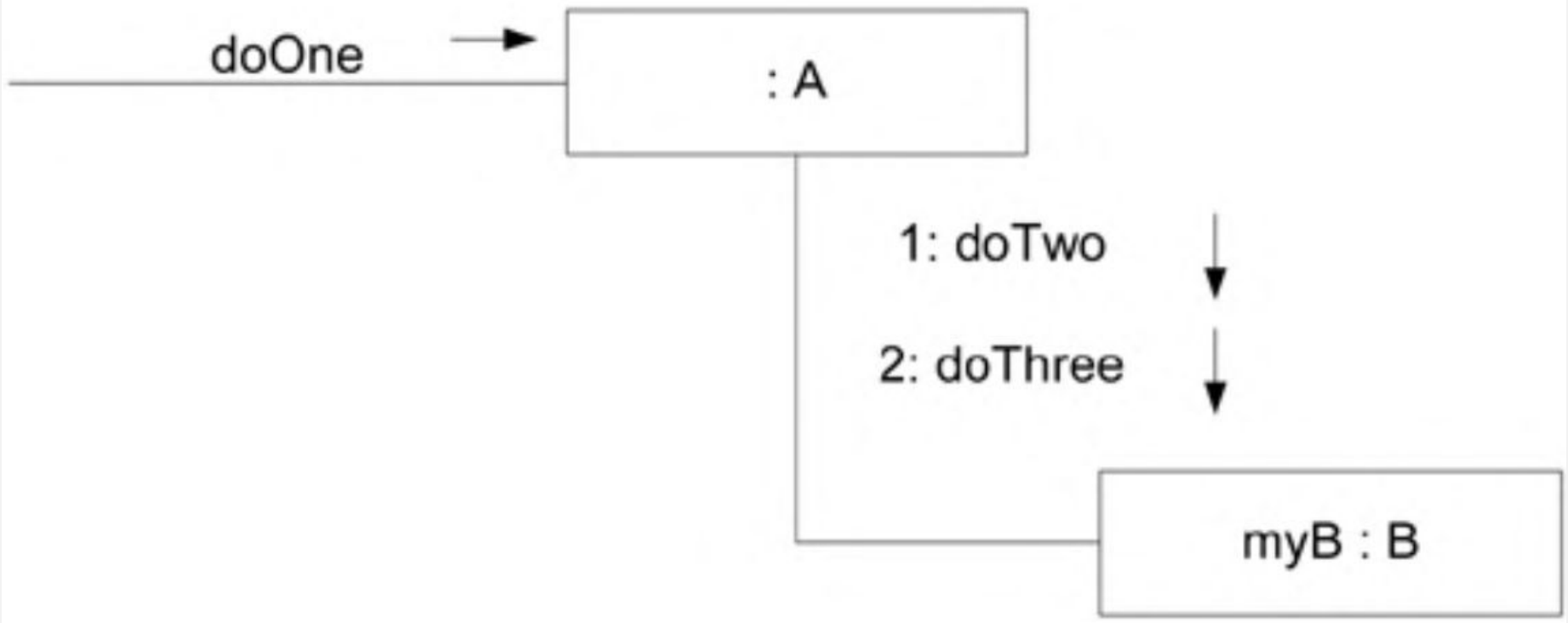
Code Representation

```
public class A
{
private B myB = new B();
public void doOne()
{
myB.doTwo();
myB.doThree();
}
// …
}
```

Code representation: Class *A* has a method named ***doOne*** and an attribute of type *B*. Also, that class *B* has methods named ***doTwo*** and ***doThree***.

Communication Diagram

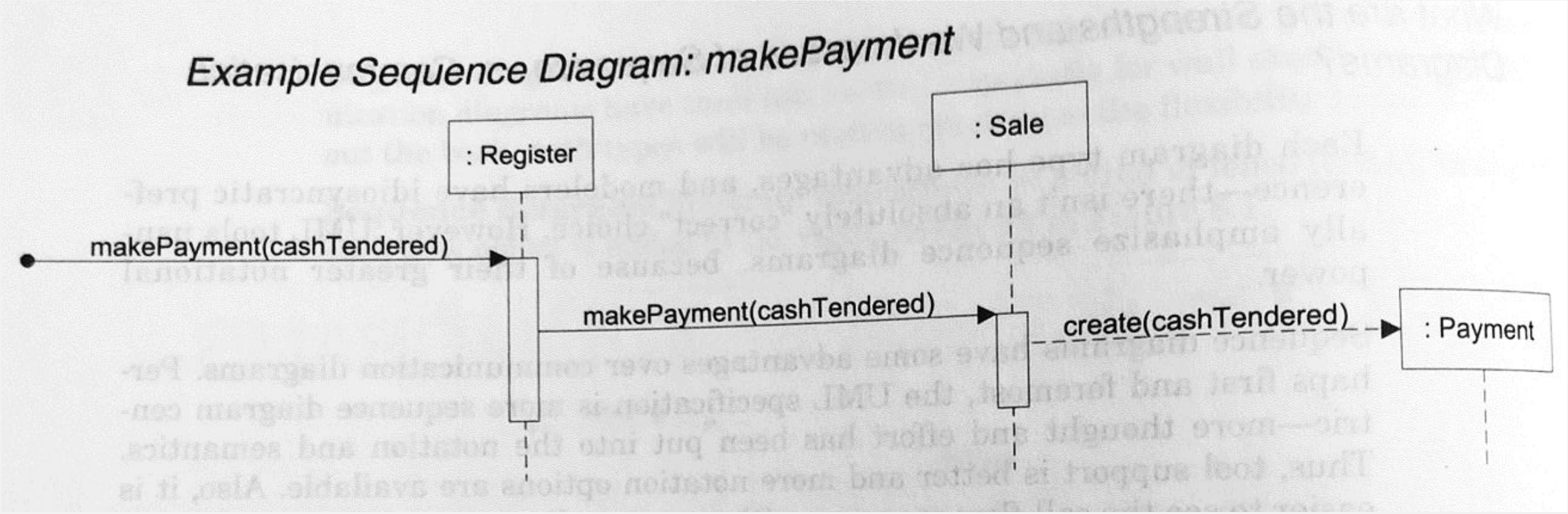
Communication diagrams illustrate object interactions in a graph or network format, in which objects can be placed anywhere on the diagram.



Strengths and Weaknesses

Type	Strengths	Weaknesses
sequence	clearly shows sequence or time ordering of messages large set of detailed notation options	forced to extend to the right when adding new objects; consumes horizontal space
communication	space economicalflexibility to add new objects in two dimensions	more difficult to see sequence of messages fewer notation options

Example Sequence Diagram: makePayment



Example Sequence Diagram: makePayment

1. The message ***makePayment*** is sent to an instance of a ***Register***. The sender is not identified.

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

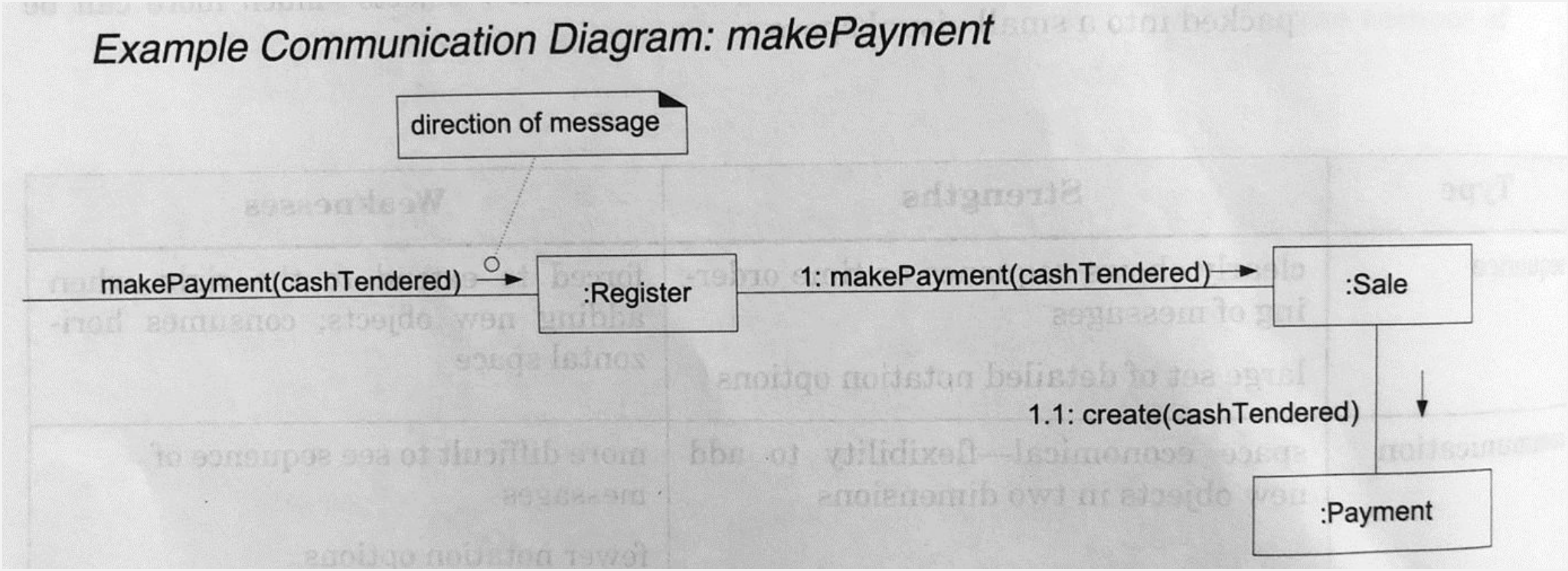
3. The ***Sale*** instance creates an instance of a ***Payment***.

Example Sequence Diagram: makePayment

Code for the ***Sale*** class and its ***makePayment*** method

```
public class Sale
{
private Payment payment;
public void makePayment( Money cashTendered )
{
payment = new Payment( cashTendered );
//…
}
// …
}
```

Example Sequence Diagram: makePayment



Common UML Interaction Diagram Notation

- ***Illustrating Participants with Lifeline Boxes***

  In the UML, the boxes you've seen in the prior sample interaction diagrams are called **lifeline** boxes.

- They represent the **participants** in the interaction.

  .

Basic Message Expression Syntax

- Interaction diagrams show messages between objects; the UML has a standard syntax for these message expressions:

```
return = message(parameter : parameterType) : returnType
```

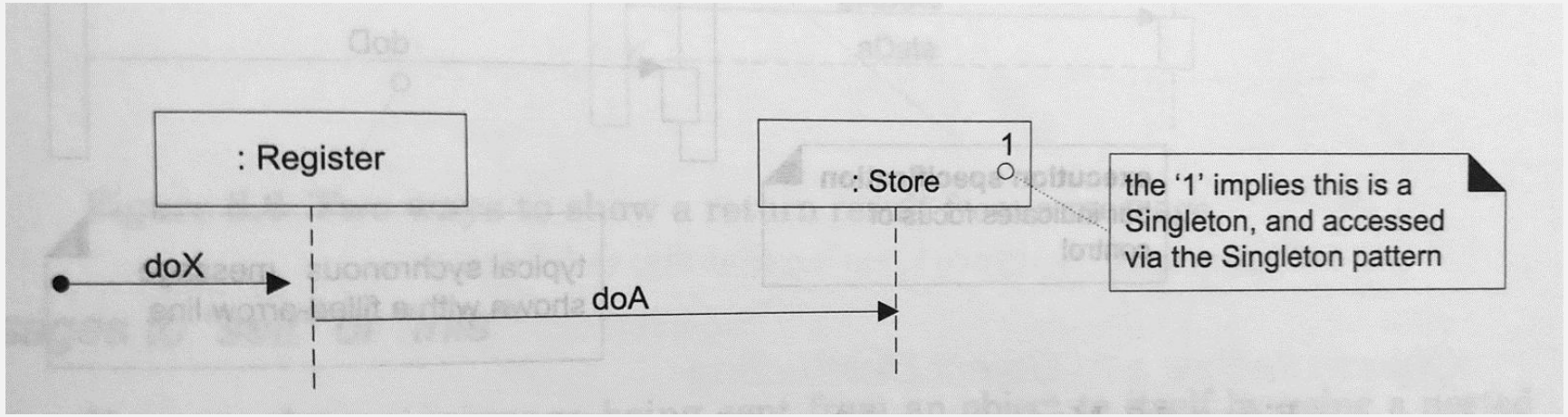
- Parentheses are usually excluded if there are no parameters, though still legal.
- Type information may be excluded if obvious or unimportant.

For example:

```
initialize(code)
initialize
d = getProductDescription(id)
d = getProductDescription(id:ItemID)
d = getProductDescription(id:ItemID) : ProductDescription
```

Basic Message Expression Syntax

- In the world of OO design patterns, there is one that is especially common, called the **Singleton** pattern.
  - There is only *one* instance of a class instantiated, never two.
  - In other words, it is a "singleton" instance.
- In a UML interaction diagram (sequence or communication), such an object is marked with a '1' in the upper right corner of the lifeline box.



Basic Sequence Diagram Notation

Lifeline Boxes and Lifelines

- In contrast to communication diagrams, in sequence diagrams the lifeline boxes include a vertical line extending below them - these are the actual lifelines.
- Although all UML examples show the lifeline as dashed (because of UML 1 influence).
- In fact the UML 2 specification says it may be solid *or* dashed.

Basic Sequence Diagram Notation

Messages

- Each (typical synchronous) message between objects is represented with a message expression on a *filled-arrowed* solid line between the vertical lifelines.
- The time ordering is organized from top to bottom of lifelines.

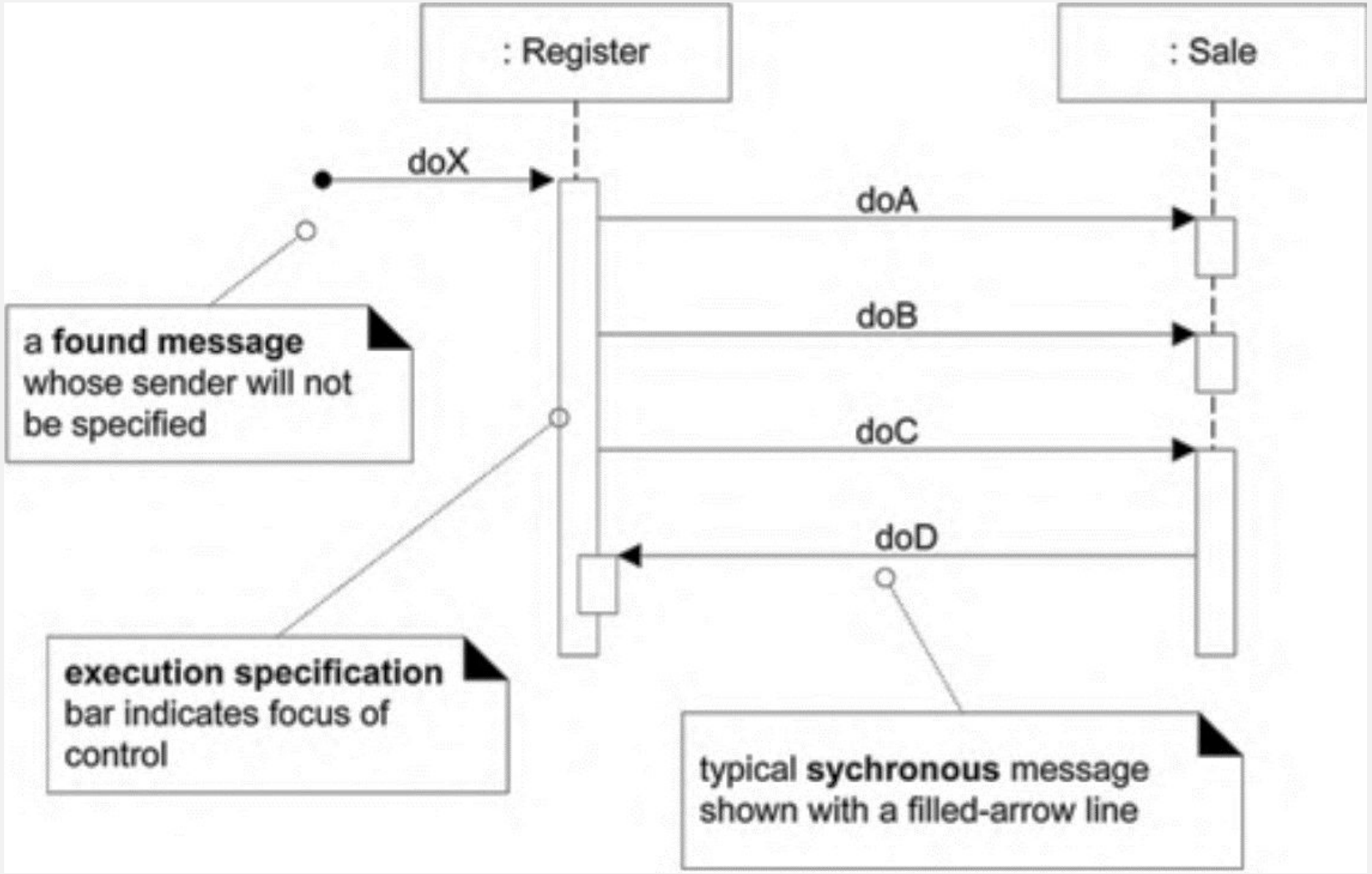
Basic Sequence Diagram Notation

Focus of Control and Execution Specification Bars

- Sequence diagrams may also show the focus of control using an **execution specification** bar (previously called an **activation bar** or simply an **activation** in UML 1).
- The bar is optional.

Basic Sequence Diagram Notation

Focus of Control and Execution Specification Bars



Basic Sequence Diagram Notation

Illustrating Reply or Returns

There are two ways to show the return result from a message:

1. Using the message syntax *returnVar = message(parameter)*.

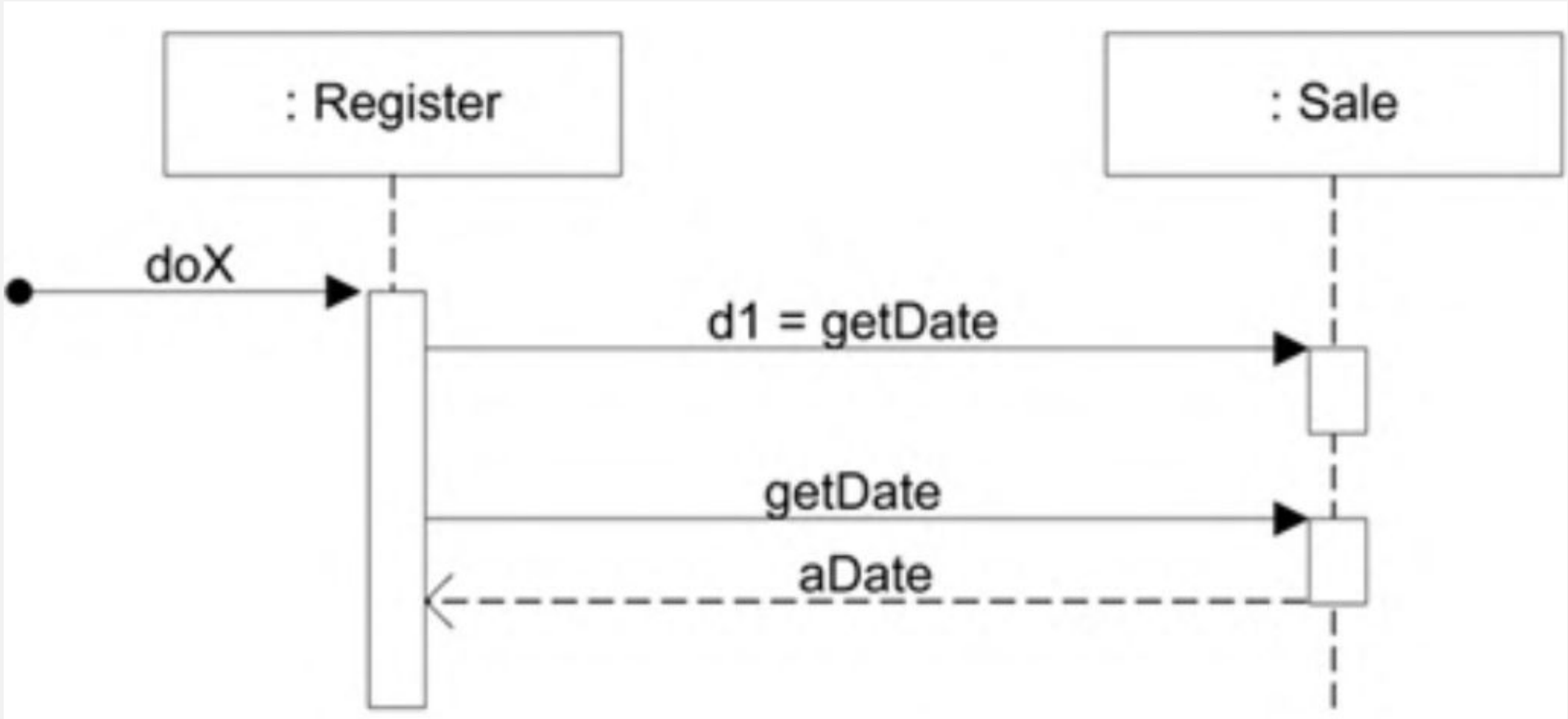
2. Using a reply (or return) message line at the end of an activation bar.

Both are common in practice.

If the reply line is used, the line is normally labelled with an arbitrary description of the returning value.

Basic Sequence Diagram Notation

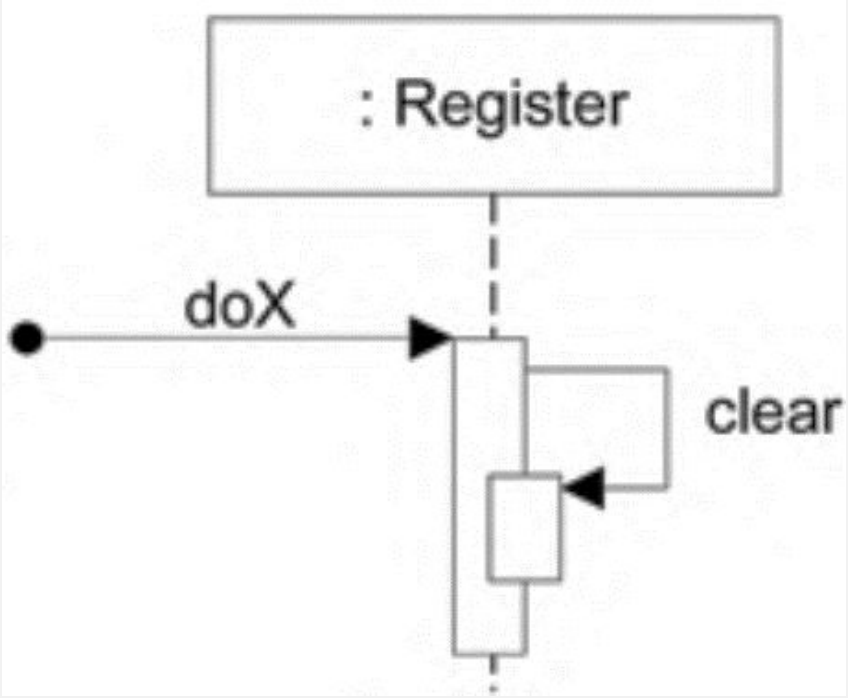
Illustrating Reply or Returns



Basic Sequence Diagram Notation

Messages to "self" or "this“

- You can show a message being sent from an object to itself by using a nested activation bar.



Basic Sequence Diagram Notation

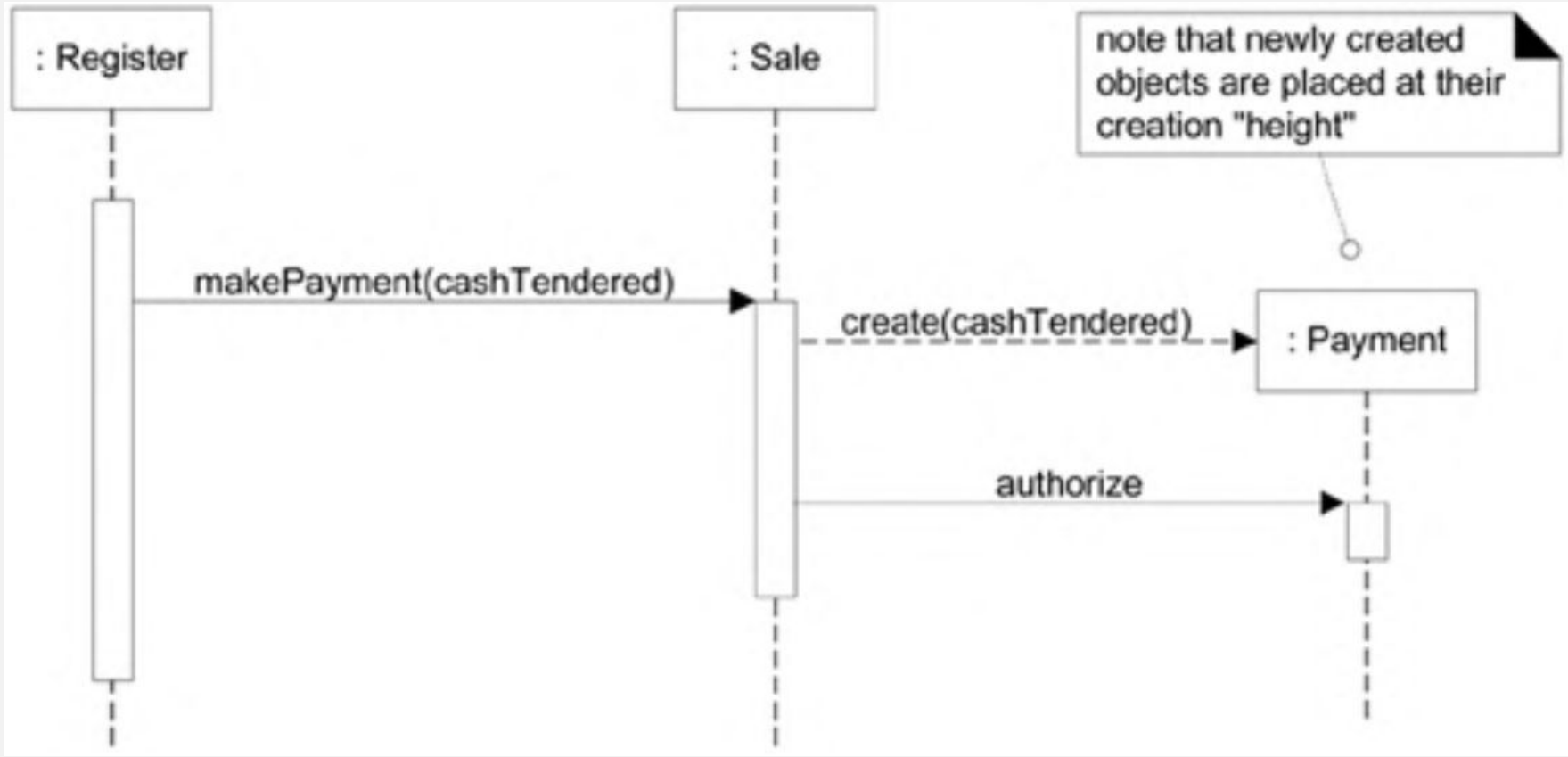
Creation of Instances

- Object creation notation is shown in below. Note the UML-mandated *dashed* line.
- The arrow is filled if it's a regular synchronous message or open (stick arrow) if an asynchronous call.
- The message name *create* is not required

.

Basic Sequence Diagram Notation

Creation of Instances



Basic Sequence Diagram Notation

Object Lifelines and Object Destruction

In some circumstances it is desirable to show explicit destruction of an object, For example,

- When using C++ which does not have automatic garbage collection,
- When you want to especially indicate an object is no longer usable (such as a closed database connection).

- The UML lifeline notation provides a way to express this destruction.

.

Basic Sequence Diagram Notation

Object Lifelines and Object Destruction

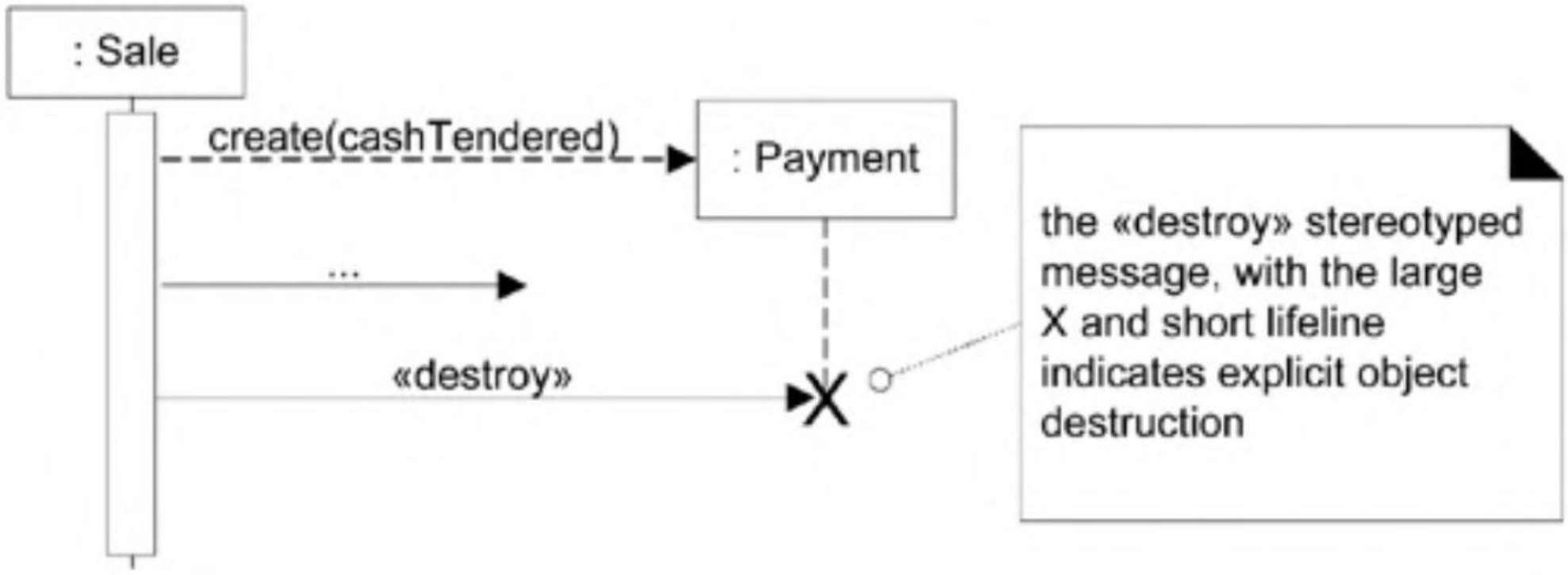


Diagram Frames in UML Sequence Diagrams

Frames

- To support conditional and looping constructs (among many other things), the UML uses **frames**.
- Frames are regions or fragments of the diagrams; they have an operator or label (such as *loop*) and a guard (conditional clause).

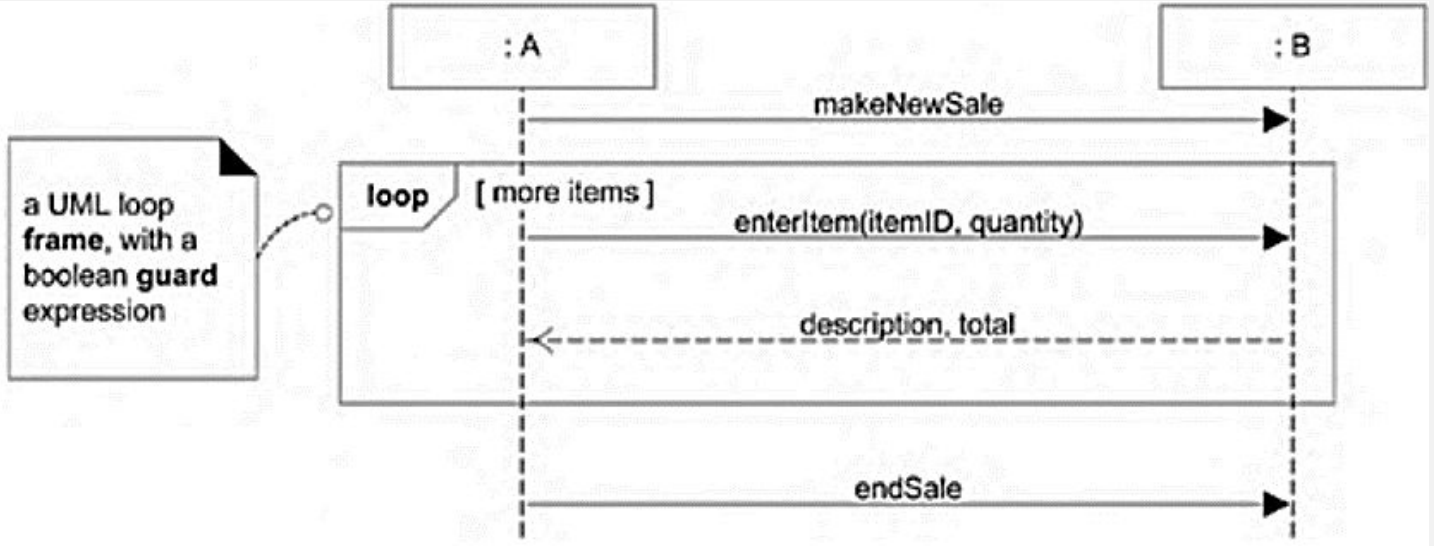


Diagram Frames in UML Sequence Diagrams

- Frame Operations:

Frame Operator	Meaning
alt	Alternative fragment for mutual exclusion conditional logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write *loop(n)* to indicate looping n times. There is discussion that the specification will be enhanced to define a *FOR* loop, such as *loop(i, 1, 10)*
opt	Optional fragment that executes if guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

Diagram Frames in UML Sequence Diagrams

Conditional Messages

An OPT frame is placed around one or more messages. Notice that the guard is placed *over* the related lifeline.

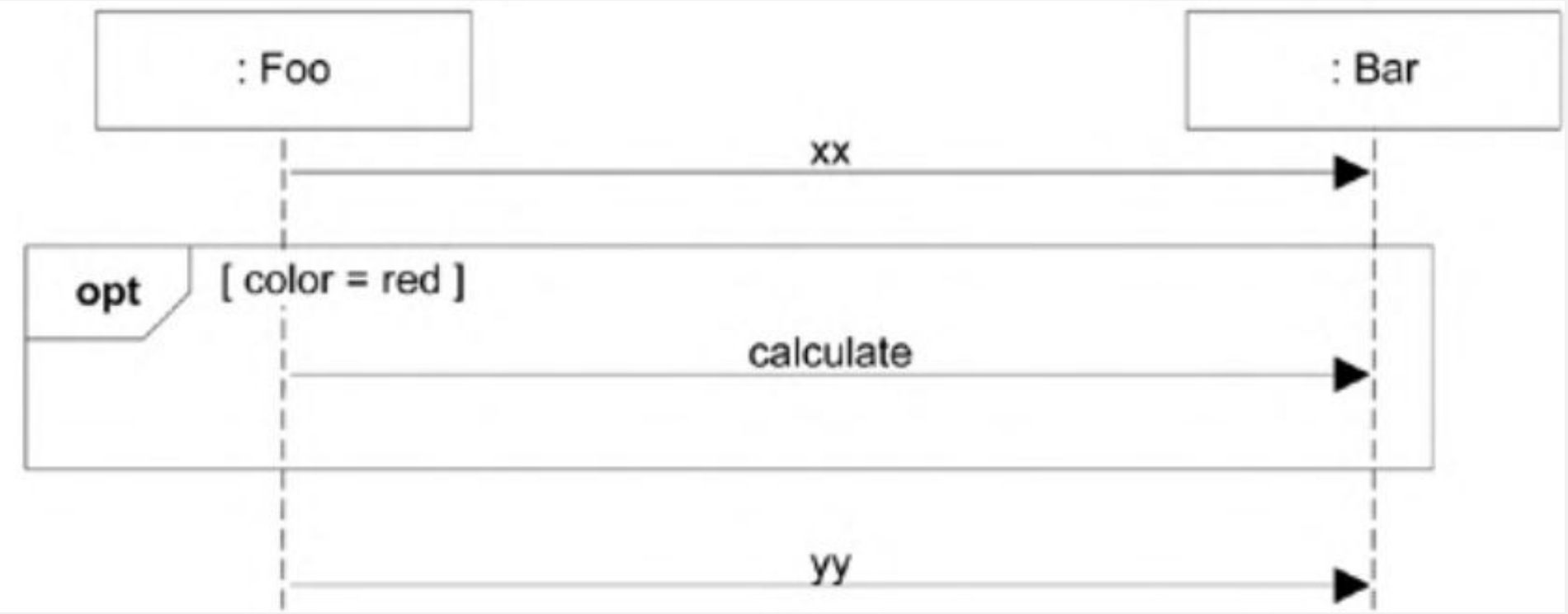
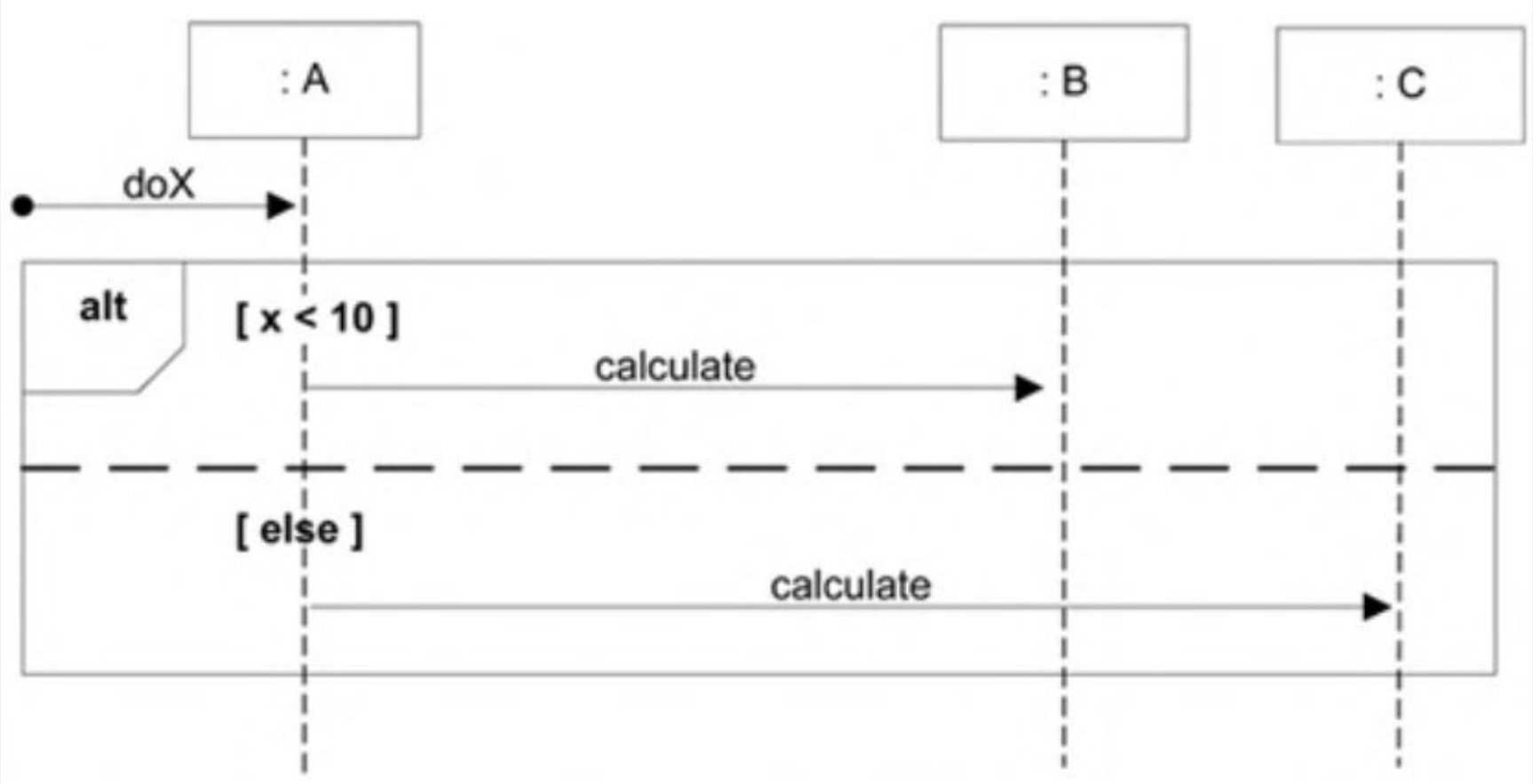
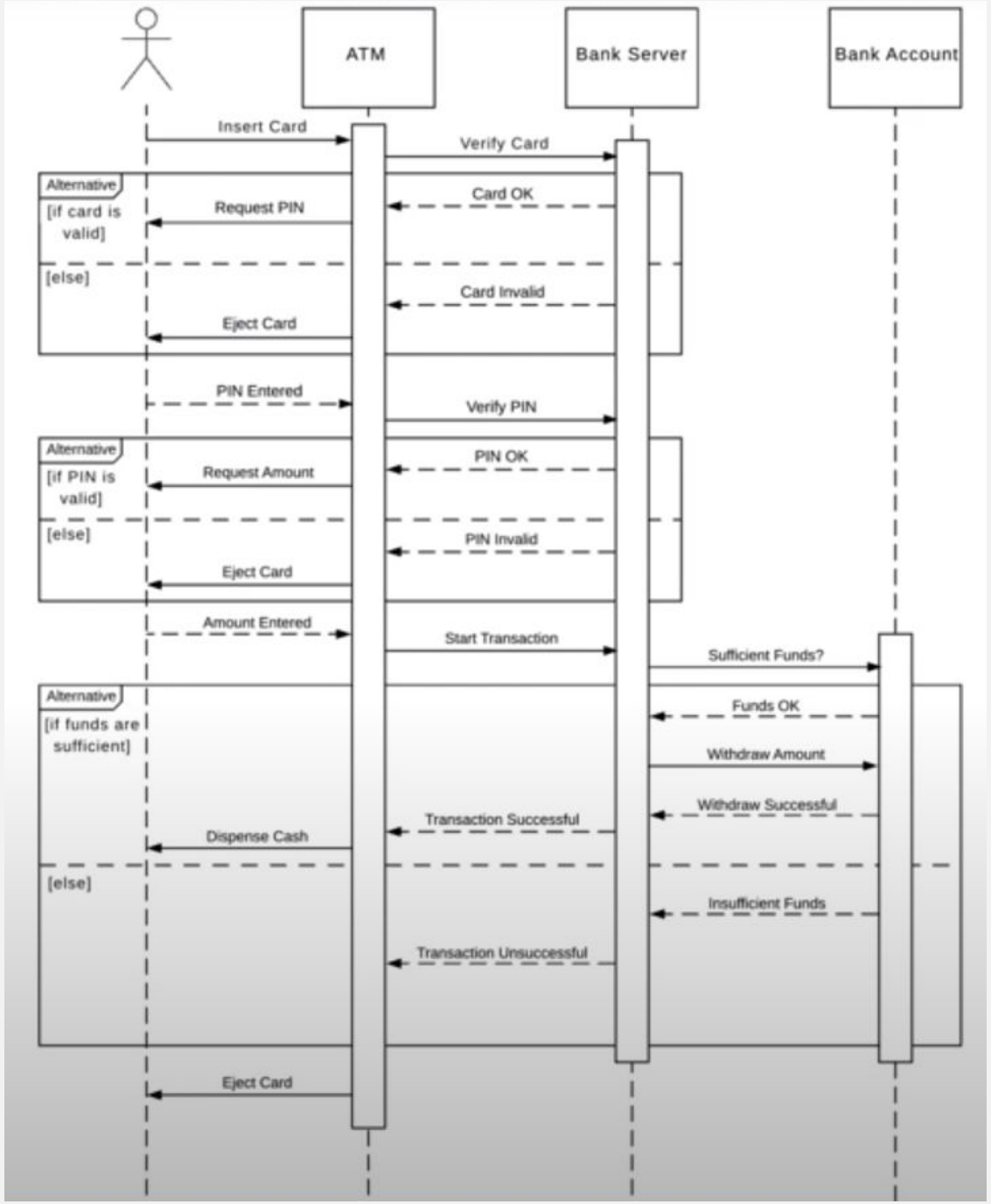


Diagram Frames in UML Sequence Diagrams

Mutually Exclusive Conditional Messages

- An ALT frame is placed around the mutually exclusive alternatives.




ATM
Bank Server
Bank Account
Insert Card
Verify Card
Alternative
[if card is valid]
Request PIN
Card OK
[else]
Card Invalid
Eject Card
PIN Entered
Verify PIN
Alternative
[if PIN is valid]
Request Amount
PIN OK
[else]
PIN Invalid
Eject Card
Amount Entered
Start Transaction
Sufficient Funds?
Alternative
[if funds are sufficient]
Funds OK
Withdraw Amount
Withdraw Successful
Transaction Successful
Dispense Cash
[else]
Insufficient Funds
Transaction Unsuccessful
Eject Card

End of Lecture

Any Question ?