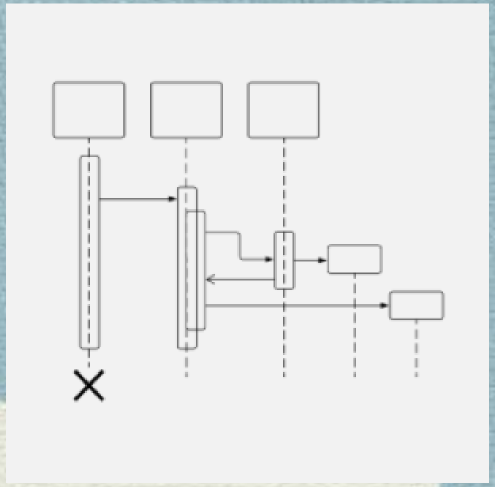


Software Modeling and Analysis
CS 284

Lab 5

Sequence Diagrams

Something is Missing!

- Use cases allow your model to describe **what your system must be able to do (functional requirements)**
- Classes allow your model to describe **the different types of parts (objects / classes)**that make up your system's structure.

Something is Missing!

- There's one large piece that's missing; with use cases and classes alone, **you can't yet model how your system is actually going to do its job.**
- This is where **interaction diagrams**, and specifically **sequence diagrams**, come into play.

In today's session you will learn ..

- What is sequence diagrams
- Where to use sequence diagrams
- Graphical notations: participants, lifelines , activation bars, and messages .

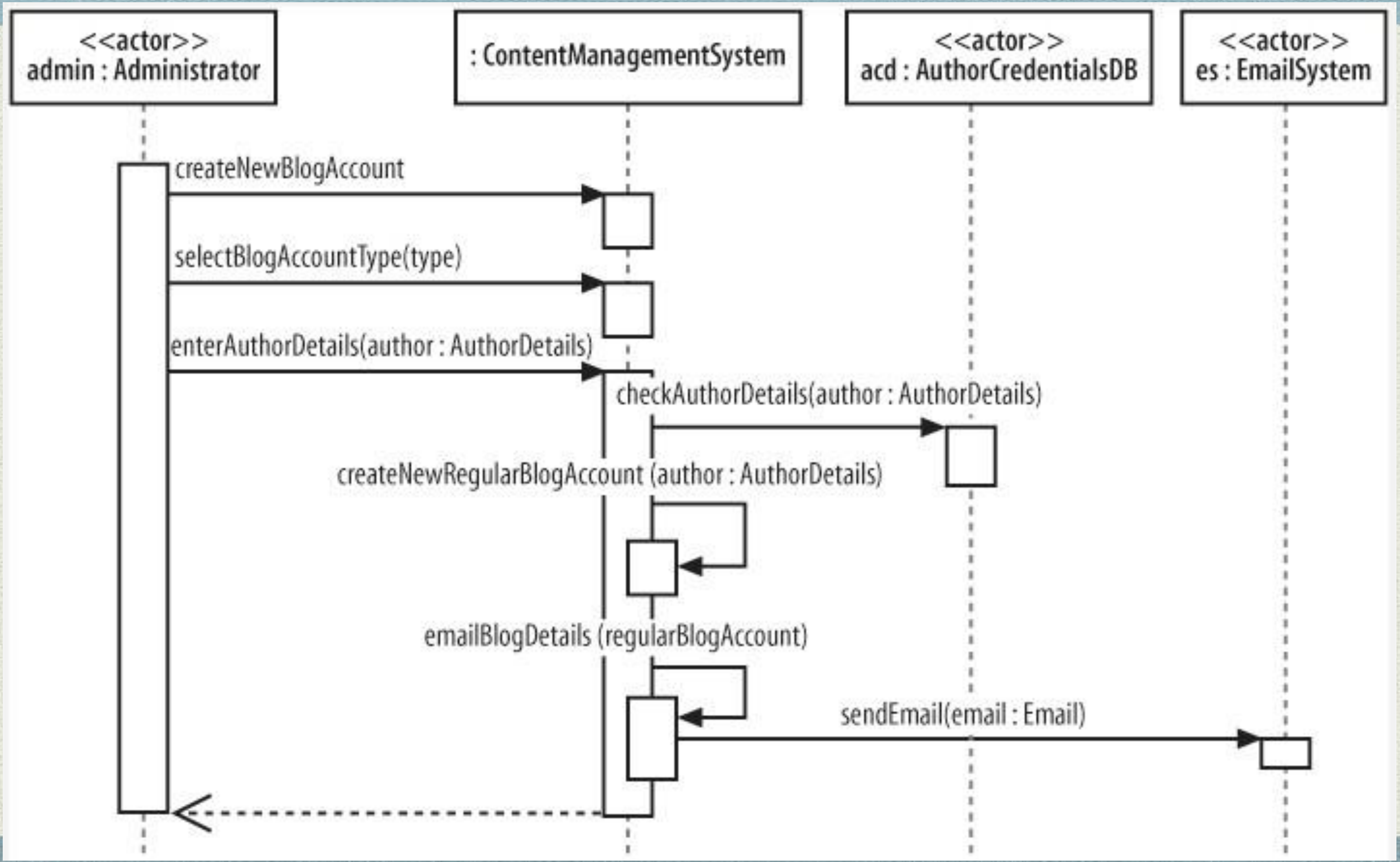
Sequence Diagrams

- **Sequence diagrams are all about capturing the order of interactions between parts of your system.**
- Using a sequence diagram, you can describe **which interactions** will be triggered when a particular use case is executed and **in what order those interactions will occur.**
- Typically, a sequence diagram captures the behavior of **a single scenario**. It **shows a number of example objects and the messages that are passed between these objects within the use case.**

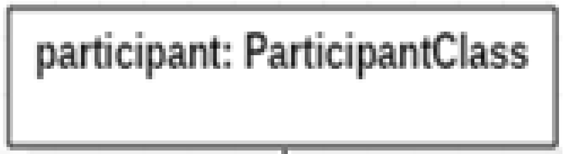
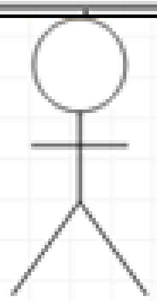
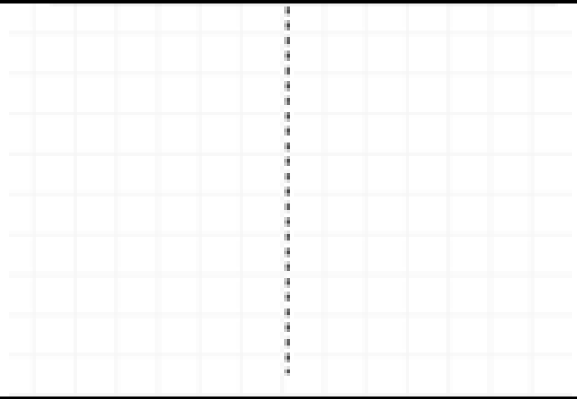
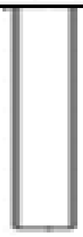
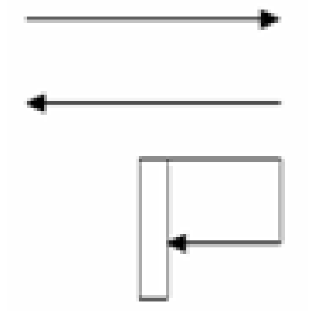
When to Use Sequence Diagrams?

- **When you want to look at the behavior of several objects within a single use case.**
- Sequence diagrams are good at showing collaborations among the objects.
- If you want to look at the behavior of a single object across many use cases, use a **state diagram.**
- If you want to look at behavior across many use cases or many threads, consider an **activity diagram.**

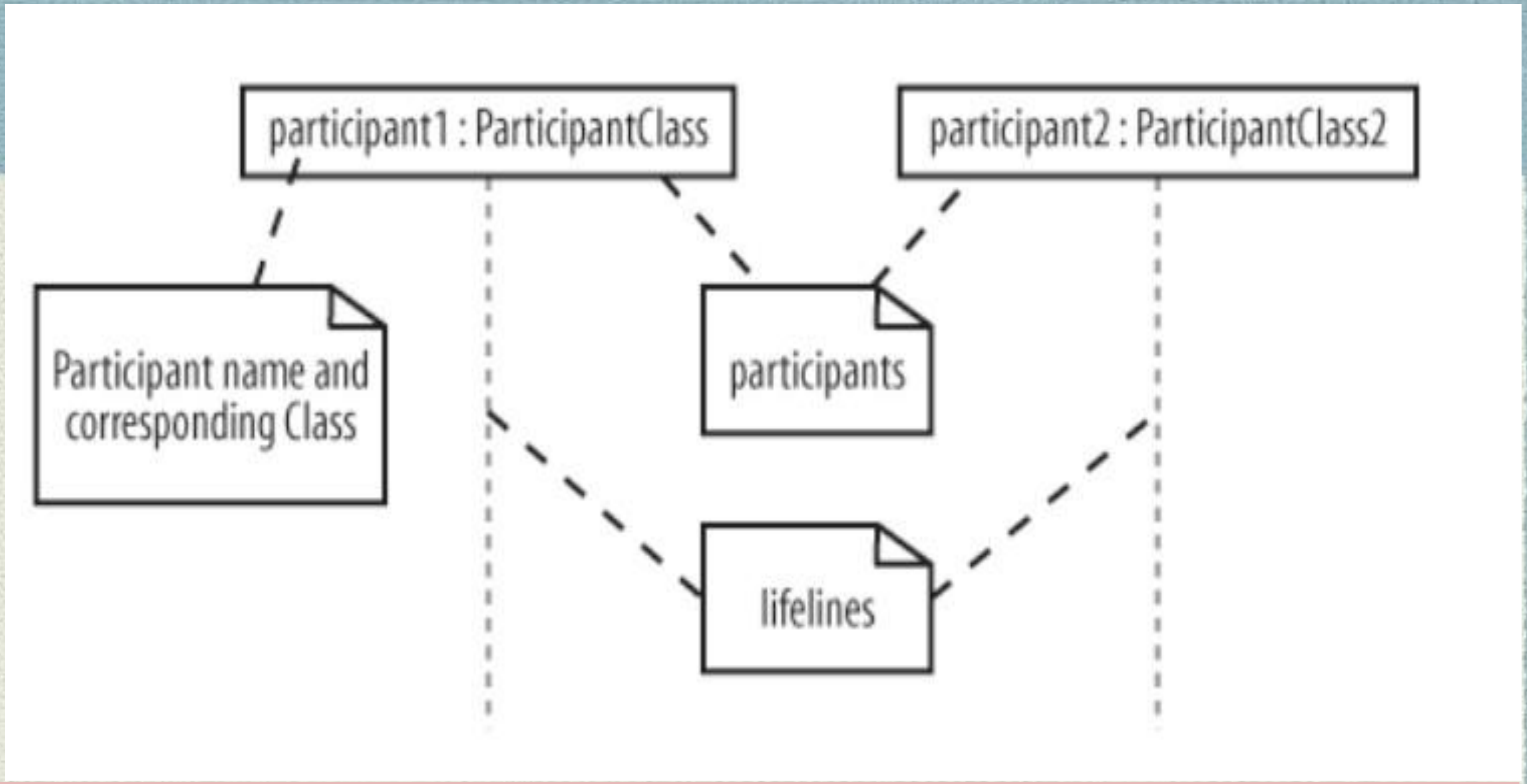
How it looks like!


<<actor>>
admin : Administrator
: ContentManagementSystem
<<actor>>
acd : AuthorCredentialsDB
<<actor>>
es : EmailSystem
createNewBlogAccount
selectBlogAccountType(type)
enterAuthorDetails(author : AuthorDetails)
checkAuthorDetails(author : AuthorDetails)
createNewRegularBlogAccount (author : AuthorDetails)
emailBlogDetails (regularBlogAccount)
sendEmail(email : Email)

Common Graphical notations of Sequence Diagram

Notation	Name
	Participant
	Actor
	Lifeline
	Activation Bar
	Message

Participants and Lifelines in a Sequence Diagram


participant1 : ParticipantClass
participant2 : ParticipantClass2
Participant name and corresponding Class
participants
lifelines

Participants

Participants are the parts of your system (usually software objects) that interact with each other during the sequence.

Lifeline

Each participant has a corresponding lifeline running down the page. A participant's lifeline shows the existence of object over a period of time.

Participant Names

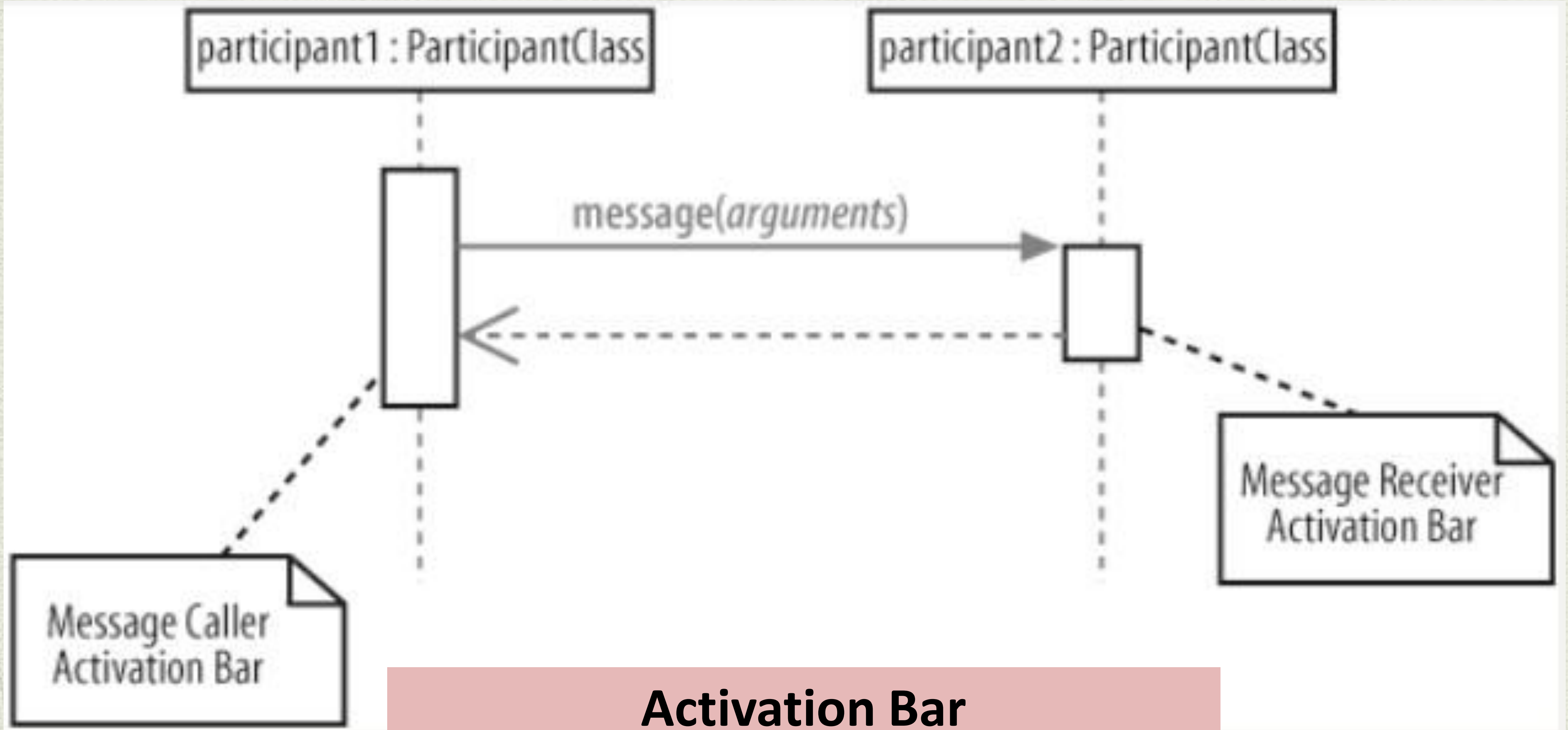
- Participants on a sequence diagram can be named in number of different ways, picking elements from the standard format:

```
name [selector] : class_name
```

- The elements of the format that you pick to use for a particular participant will depend on the information known about a participant at a given time.

admin	A part is named `admin`, but at this point in time the part has not been assigned a class.
: ContentManagementSystem	The class of the participant is `ContentManagementSystem`, but the part currently does not have its own name.
admin : Administrator	There is a part that has a name of `admin` and is of the class `Administrator`.
eventHandlers [2] : EventHandler	There is a part that is accessed within an array at element 2, and it is of the class `EventHandler`.

Activation Bars


participant1 : ParticipantClass
participant2 : ParticipantClass
message(*arguments*)
Message Receiver Activation Bar
Message Caller Activation Bar

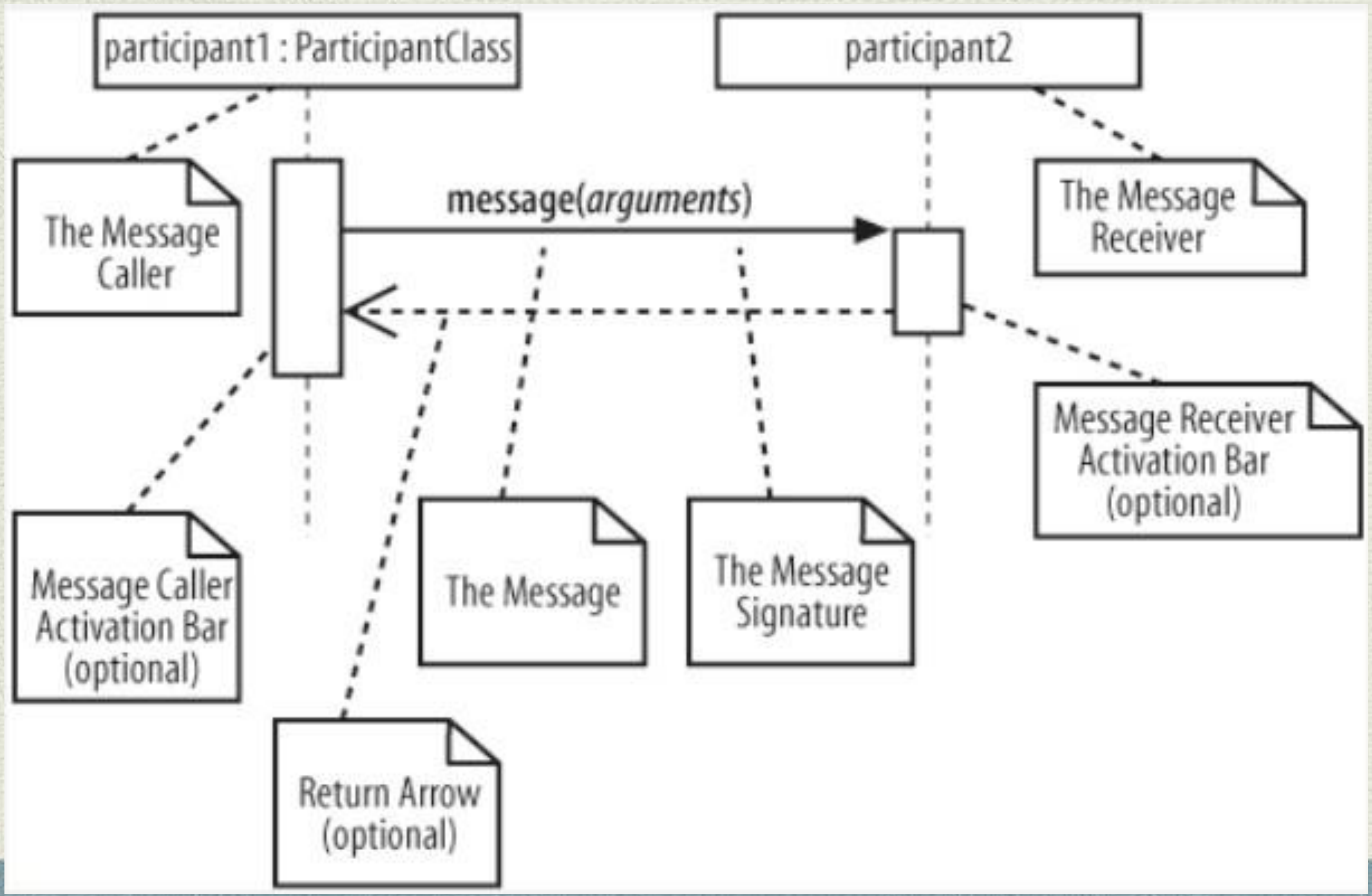
Activation Bar
Shows the period of time that an object is performing an action.

Activation Bars (Cont.)

- When a message is passed to a participant it triggers, or invokes, the receiving participant into doing something; at this point, the receiving participant is said to be active. **To show that a participant is active, i.e., doing something, you can use an activation bar.**
- An activation bar can also be shown on the sending end. It indicates that the sending participant is busy while it sends the message.
- Activation bars are **optional**—they can clutter up a diagram.

Messages in Sequence Diagrams

- An interaction in a sequence diagram occurs when one participant decides to send a message to another participant.



Messages in Sequence Diagrams (Cont.)

- **How to specify a message in sequence diagram :**

  Messages on a sequence diagram are specified using an **arrow** from **the participant that wants to pass the message, the Message Caller,** to **the participant that is to receive the message, the Message Receiver.**

- **Direction of messages:**

  Messages can flow in whatever direction makes sense for the required interaction—from left to right, right to left, or even back to the Message Caller itself.

- Think of a message as an **event** that is passed from a Message Caller to get the Message Receiver to do something

Message Signatures

- A message arrow comes with a description, or signature. The format for a message signature is:

```
attribute = signal_or_message_name (arguments) : return_type
```

- You can specify any number of different arguments on a message, each separated using a comma. The format of an argument is:

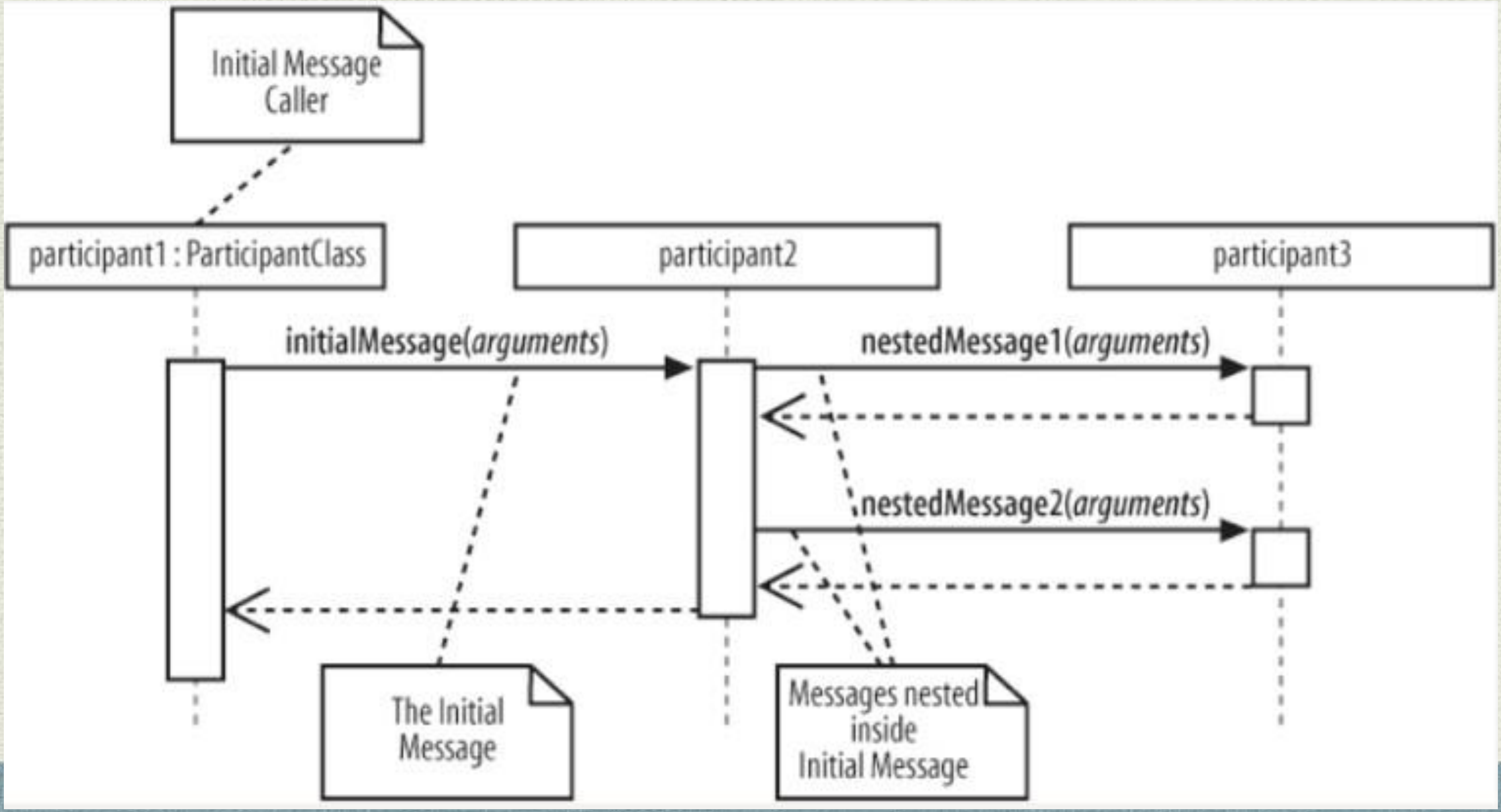
```
<name>:<class>
```

Message Signature – Examples

- doSomething( )
- doSomething(number1 : Number, number2 : Number)
- doSomething( ) : ReturnClass
- myVar = doSomething( ) : ReturnClass

Nested Messages

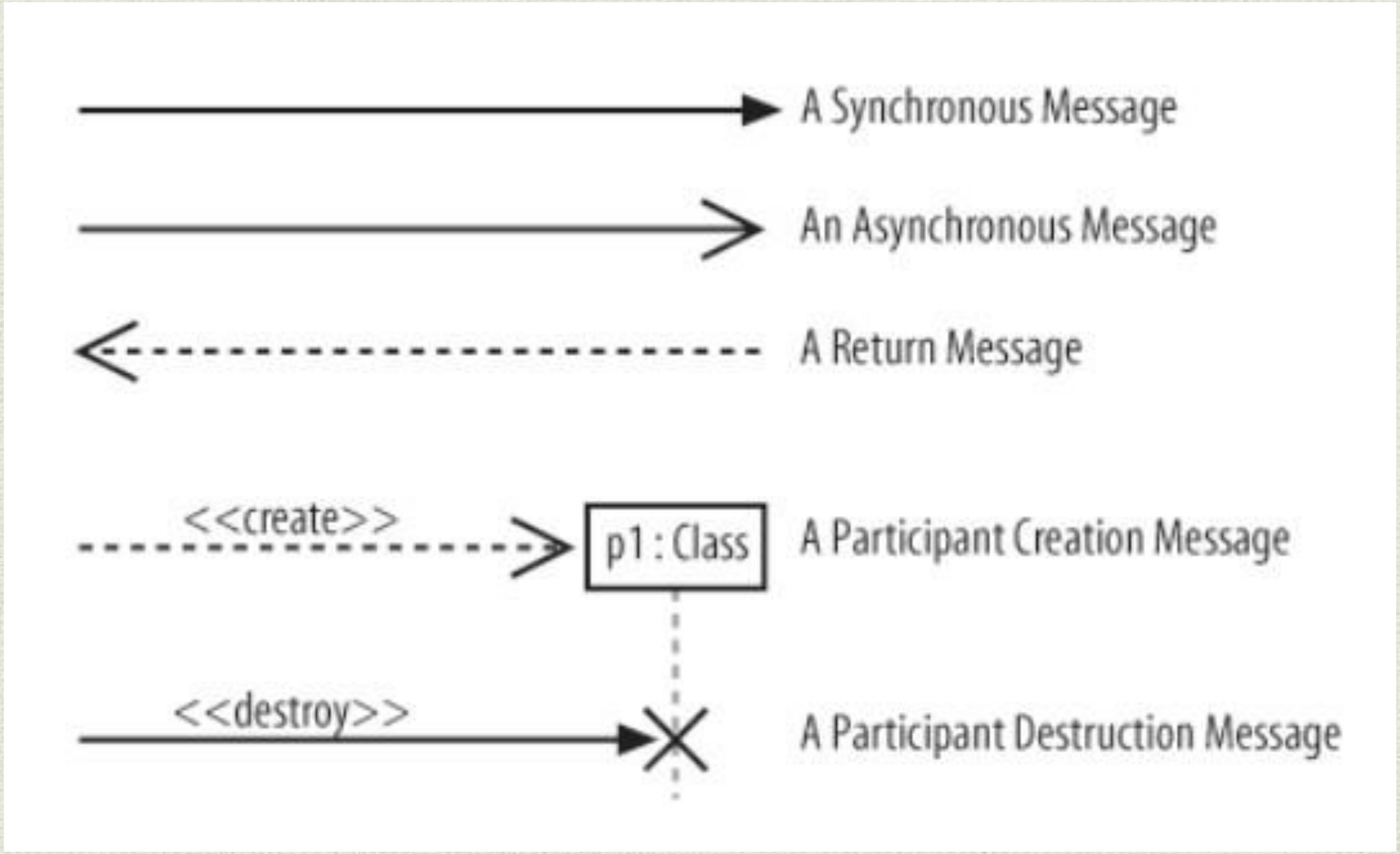
- When a message from one participant results in one or more messages being sent by the receiving participant, those resulting messages are said to be nested within the triggering message


Initial Message Caller
participant1 : ParticipantClass
participant2
participant3
initialMessage(*arguments*)
nestedMessage1(*arguments*)
nestedMessage2(*arguments*)
The Initial Message
Messages nested inside Initial Message

Message Arrows

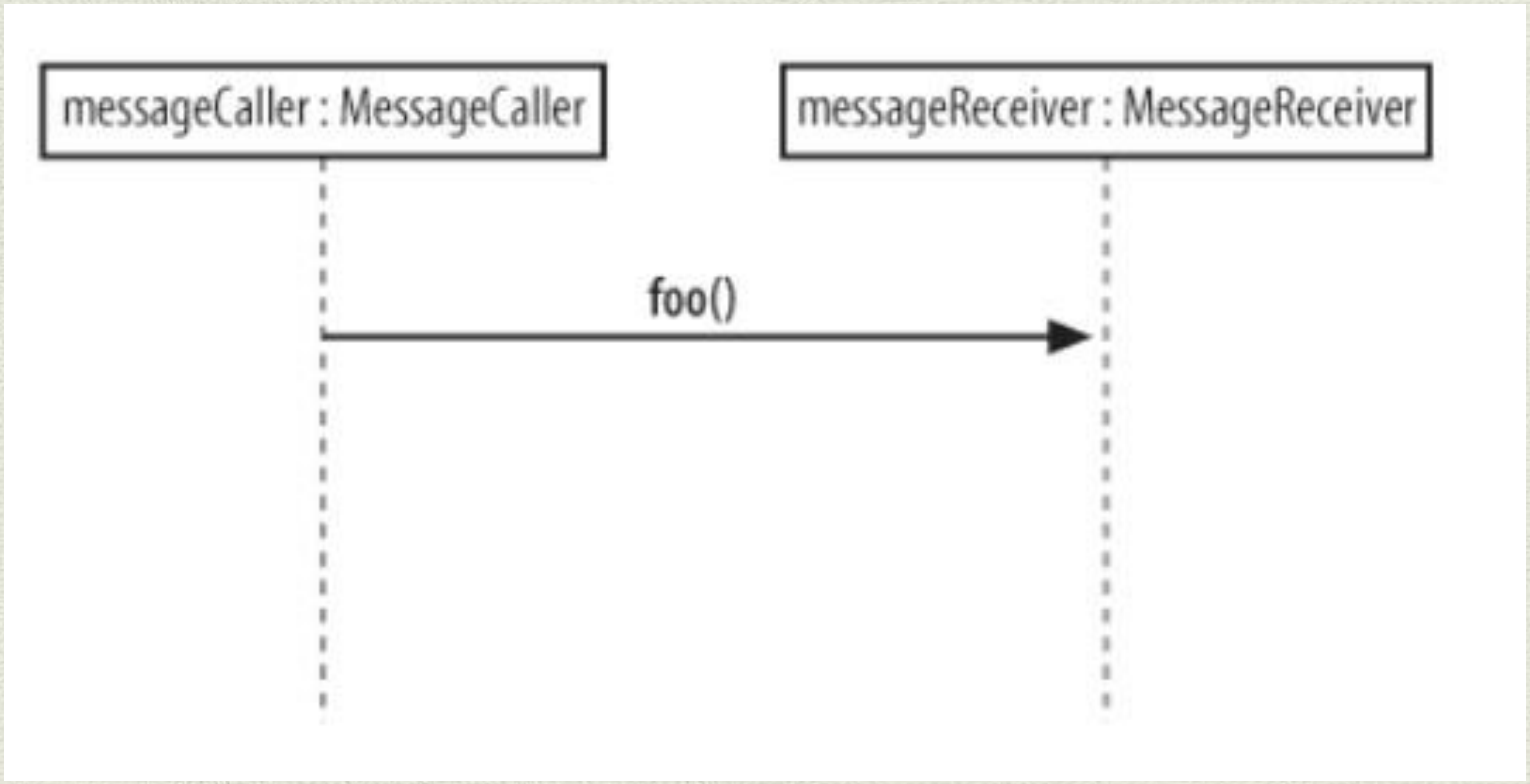
- **The type of arrowhead that is on a message is also important when understanding what type of message is being passed.**
- For example, the Message Caller may want to wait for a message to return before carrying on with its work—**a synchronous message.**
- Or it may wish to just send the message to the Message Receiver without waiting for any return as a form of "*fire and forget*" message —**an asynchronous message.**
- **Sequence diagrams need to show these different types of message using various message arrows.**

Message Arrows (Cont.)



Synchronous Messages

- A synchronous message is invoked when the Message Caller waits for the Message Receiver to return from the message invocation


messageCaller : MessageCaller
messageReceiver : MessageReceiver
foo()

Asynchronous Messages

- An asynchronous message is invoked by a Message Caller on a Message Receiver, but the Message Caller does not wait for the message invocation to return before carrying on with the rest of the interaction's steps.
- This means that the Message Caller will invoke a message on the Message Receiver and the Message Caller will be busy invoking further messages before the original message returns.

Asynchronous Messages– Example

- If you are designing a piece of software with a user interface that supports the **editing** and **printing** of a set of documents. Your application offers a button for the user to print a document.
- Printing could take some time, so you want to show that after the print button is pressed and the document is printing, the user can go ahead and work with other things in the application. Here you need a new type of message arrow: the asynchronous message arrow.

The Return Message

- **The return message** is an optional piece of notation that you can use at the end of an activation bar to show that the control flow of the activation returns to the participant that passed the original message.
- In code, a return arrow is similar to reaching the end of a method or explicitly calling a return statement.

The Return Message

- You don't have to use return messages —sometimes they can really make your sequence diagram too busy and confusing.
- Besides that there is an implied return arrow on any activation bars that are invoked using a synchronous message.

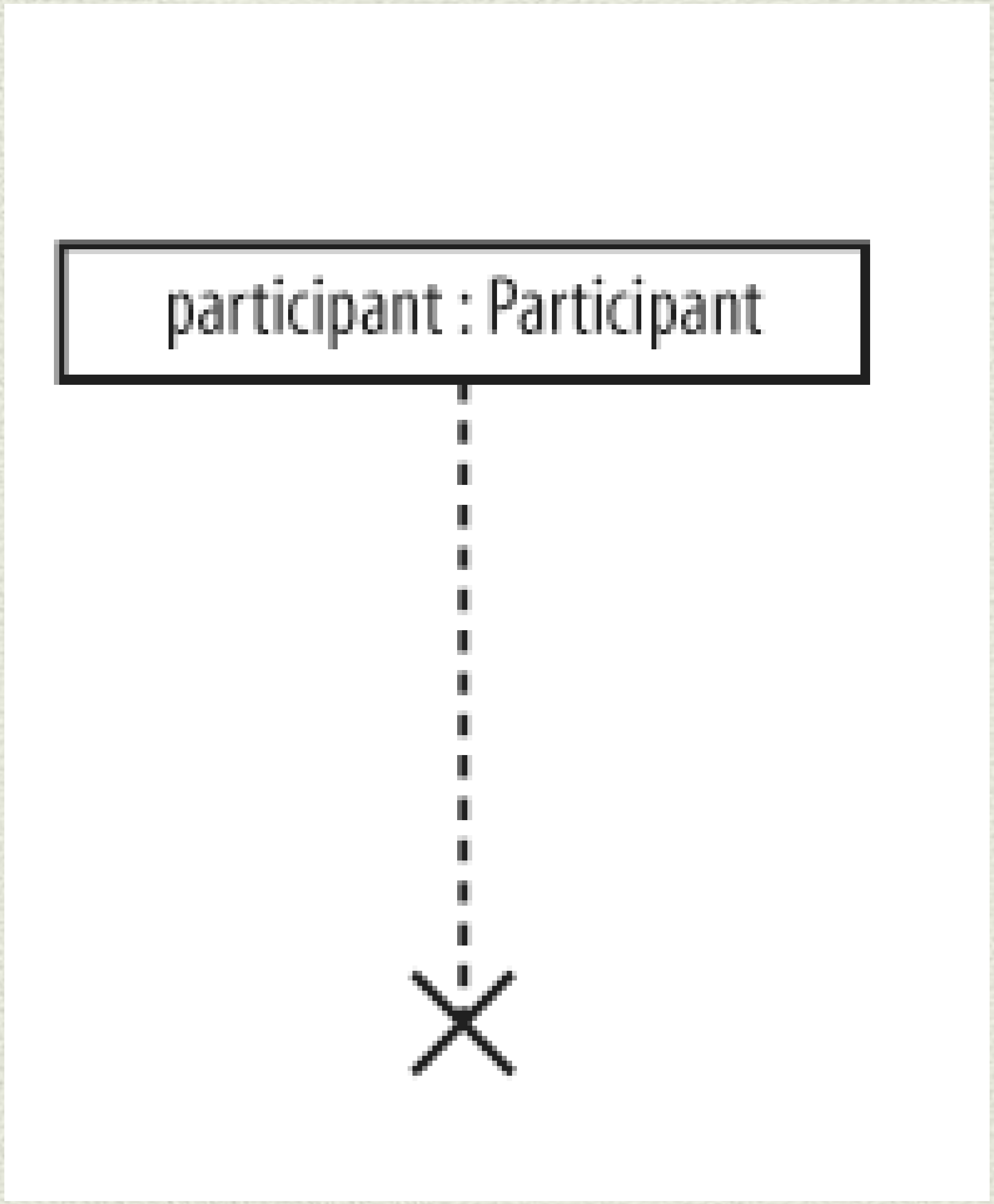
Participant Creation and Destruction Messages

- Participants do not necessarily live for the entire duration of a sequence diagram's interaction. **Participants can be created and destroyed according to the messages that are being passed**

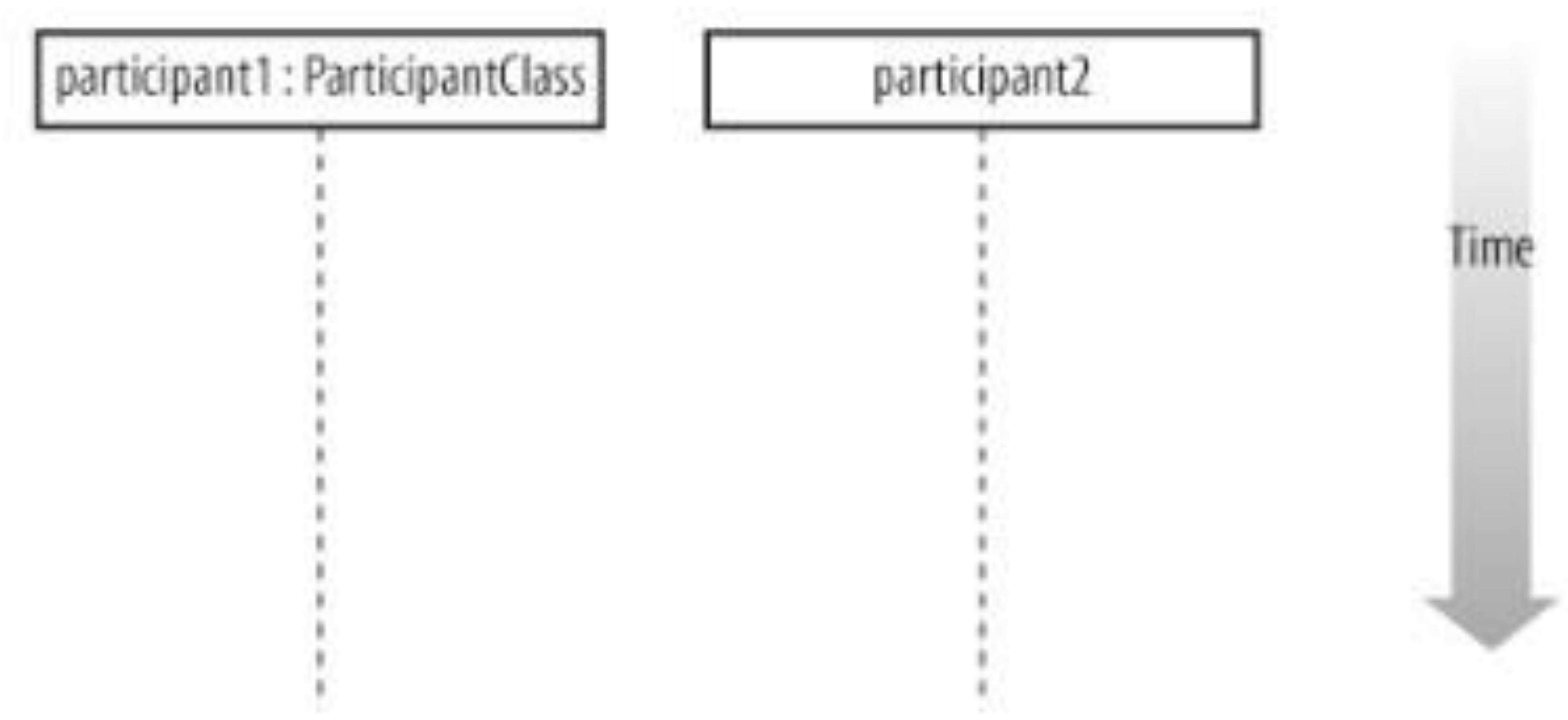
Participant Creation and Destruction Messages (Cont.)

- With some implementation languages, such as Java, you will not have an explicit destroy method so it doesn't make sense to show one on your sequence diagrams.
- It is all handled implicitly by the Java garbage collector.
- In these cases, where another factor such as the garbage collector is involved, you can either leave the object as alive but unused or imply that it is no longer needed by using the destruction cross X without an associated `destroy` method.

Participant Creation and Destruction Messages (Cont.)



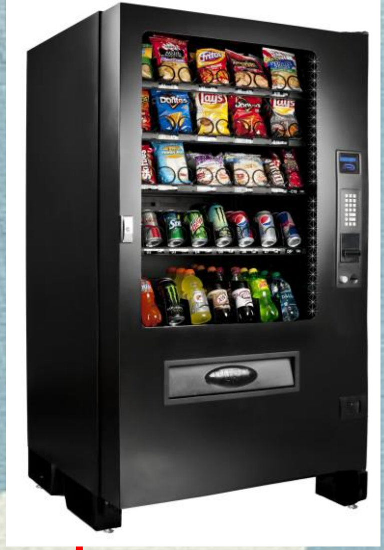
Time in Sequence Diagrams



Time in Sequence Diagrams (Cont.)

- **A sequence diagram describes the order in which the interactions take place, so time is an important factor.**
- Time on a sequence diagram starts at the top of the page, just beneath the topmost participant heading, and then progresses down the page.
- The order that interactions are placed down the page on a sequence diagram indicates the order in which those interactions will take place in time.
- **Time on a sequence diagram is all about ordering, not duration.**

Exercise 1

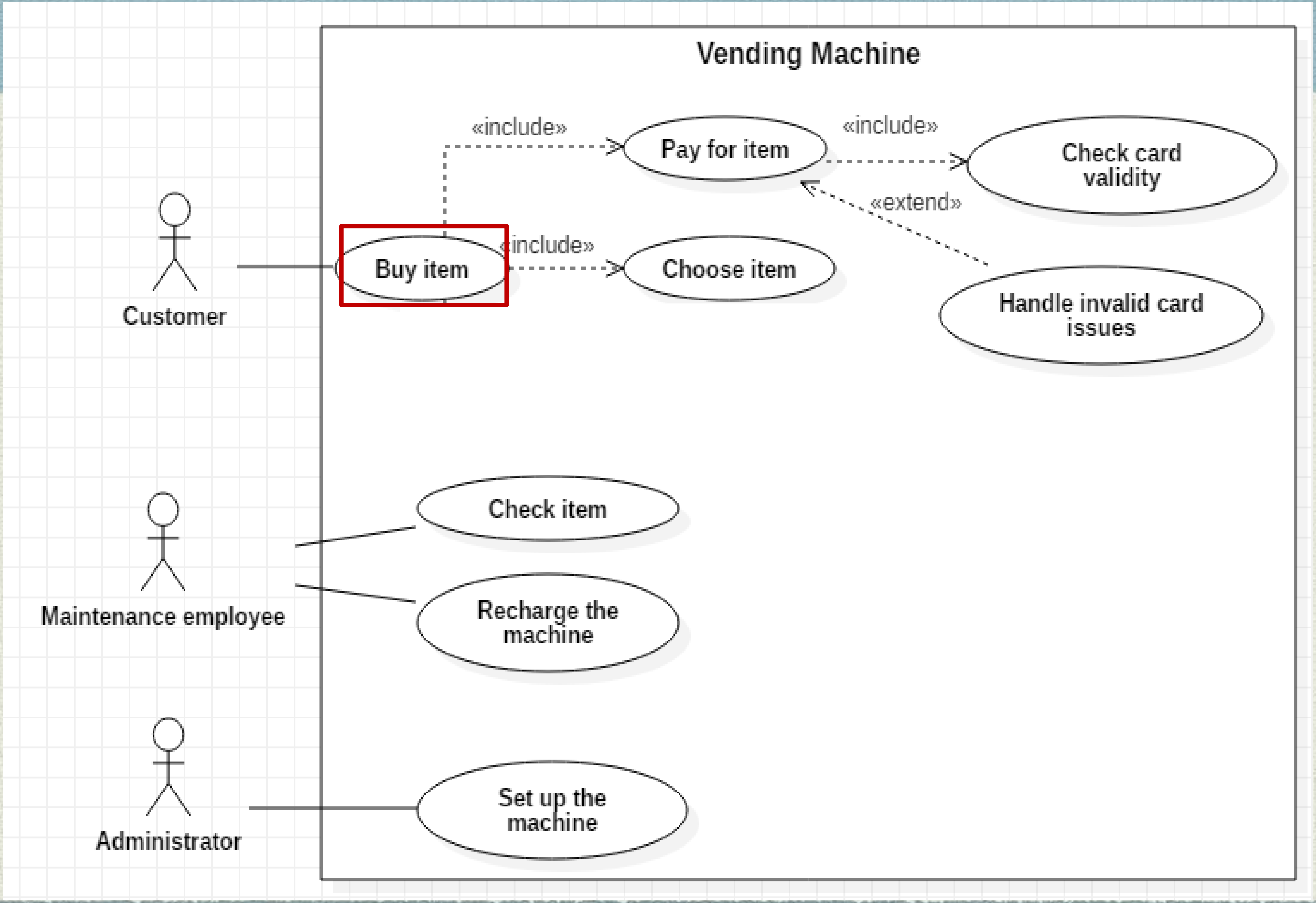


Suppose you have a customer who wants to buy an item from a vending machine using a smart card.

Vending Machine.
A vending machine sells small, packaged, ready to eat items (chocolate bars, cookies, candies, etc.). Each item has a price and a name. A customer can buy an item, using a smart card (issued by the vending machine company). No other payment method (i.e. cash, credit card) are allowed. The vending machine has a smart card reader that updates the available amount of money on the smart card upon completing the sale process. Each time the customer enters the card, the card is validated for purchase, if it is valid, the process sale will be accomplished, otherwise the card and the process will be rejected.
The system supports the following functions:
- Sell an item (choose from a list of items, pay for an item, dispense an item)
- Recharge the machine with items
- Monitor the machine (number of sold items, number of sold items per type, total revenue)
- Set up the machine (define sold items and price of items)

The system can be used by a customer (who buys item),a maintenance employee (who recharges and monitors the machines), an administrator (who sets up the machine).

1-Use case diagram:

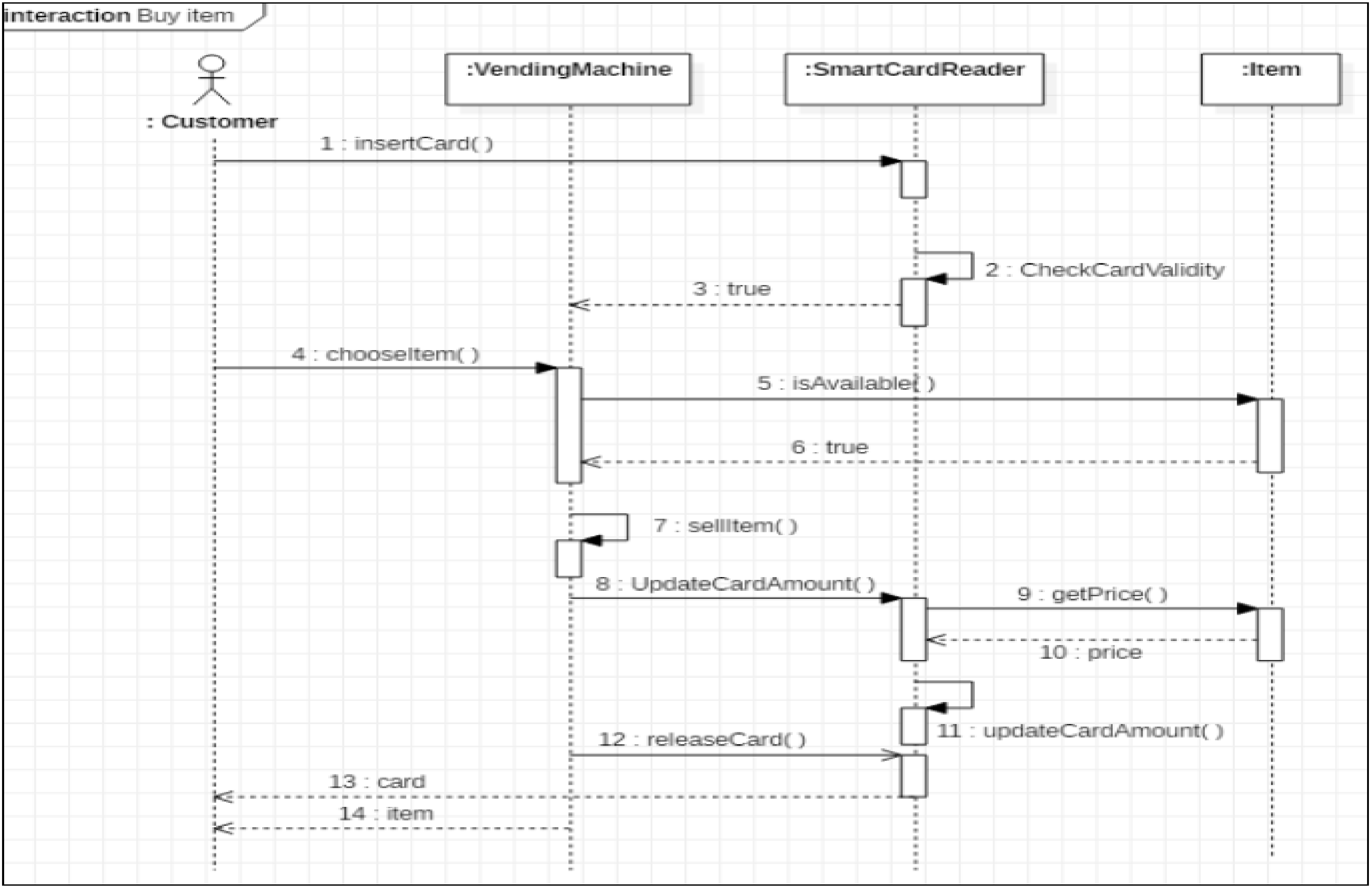


The Scenario

Suppose the following flow of events for "buy an item" use case:

1. The customer inserts a card to the smart card reader.
2. The smart card reader checks the card validity (is the card valid or not).
3. The smart card reader responds to customer "The card is valid".
4. The customer selects an item from the vending machine by typing its code.
5. The vending machine checks the item availability.
6. The vending machine get a response that "the item is available".
7. The vending machine issues an order to sell the item.
8. The vending machine asks the smart card reader to update the card amount.
9. Then the smart card reader asks for the price of the selected item.
10. The smart card reader gets the price.
11. Then the smart card updates the card amount.
12. The vending machine asks the smart card reader to release the card.
13. The smart card reader releases the card to the customer.
14. The vending machine dispenses the item to the customer.

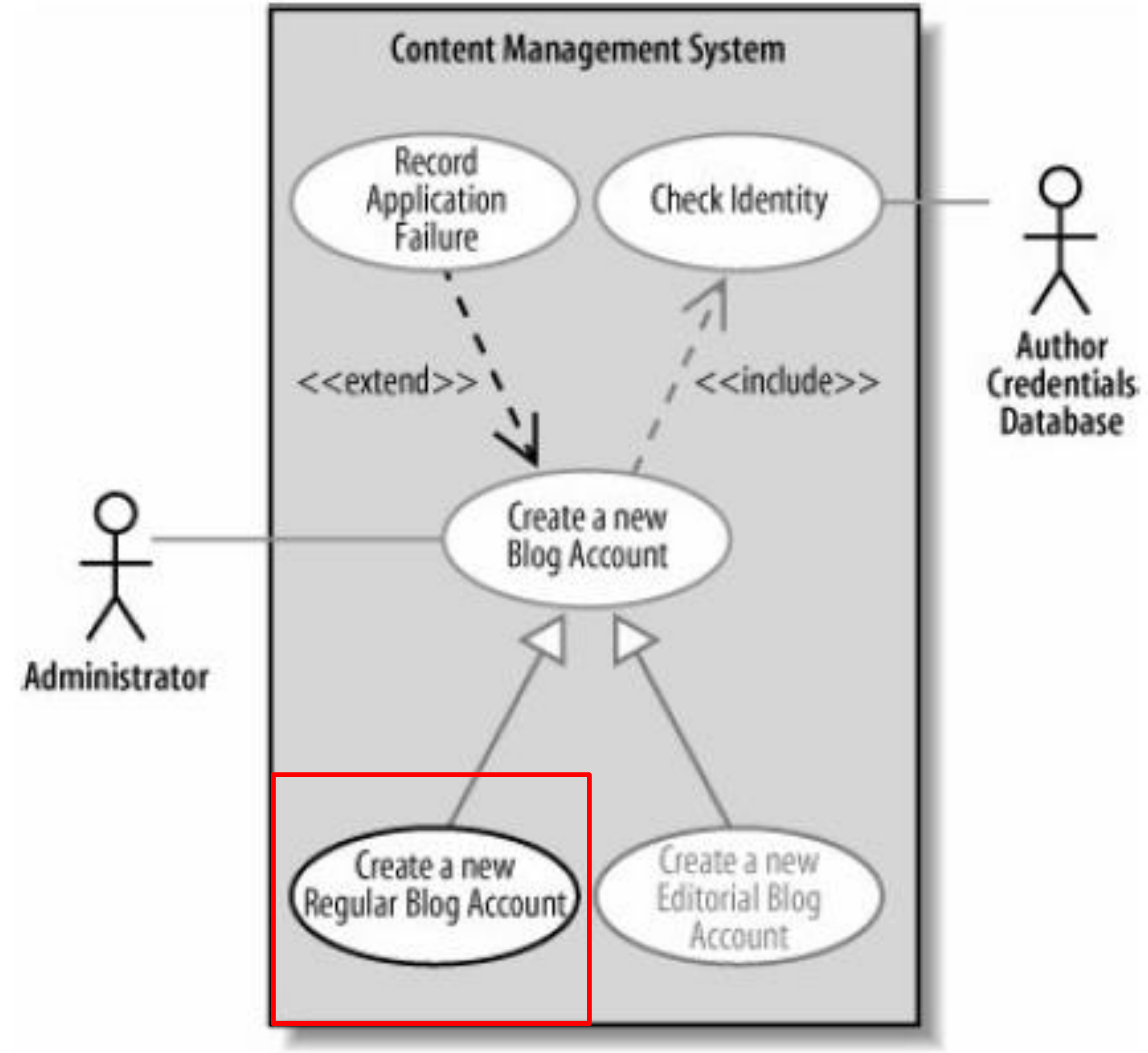
2- Sequence diagram



Bringing a Use Case to Life with a Sequence Diagram

Example 2

Figure 7-13. The Create a new Regular Blog Account use case diagram


Content Management System
Record Application Failure
Check Identity
<<extend>>
<<include>>
Author Credentials Database
Create a new Blog Account
Administrator
Create a new Regular Blog Account
Create a new Editorial Blog Account

The Scenario

1	The Administrator asks the system to create a new blog account.
2	The Administrator selects the regular blog account type.
3	The Administrator enters the author's details.
4	The author's details are checked using the Author Credentials Database.
5	The new regular blog account is created.
6	A summary of the new blog account's details are emailed to the author.

Figure 7-14. This sequence diagram shows the actors that interact with your system and your system is shown simply as a single part in the sequence

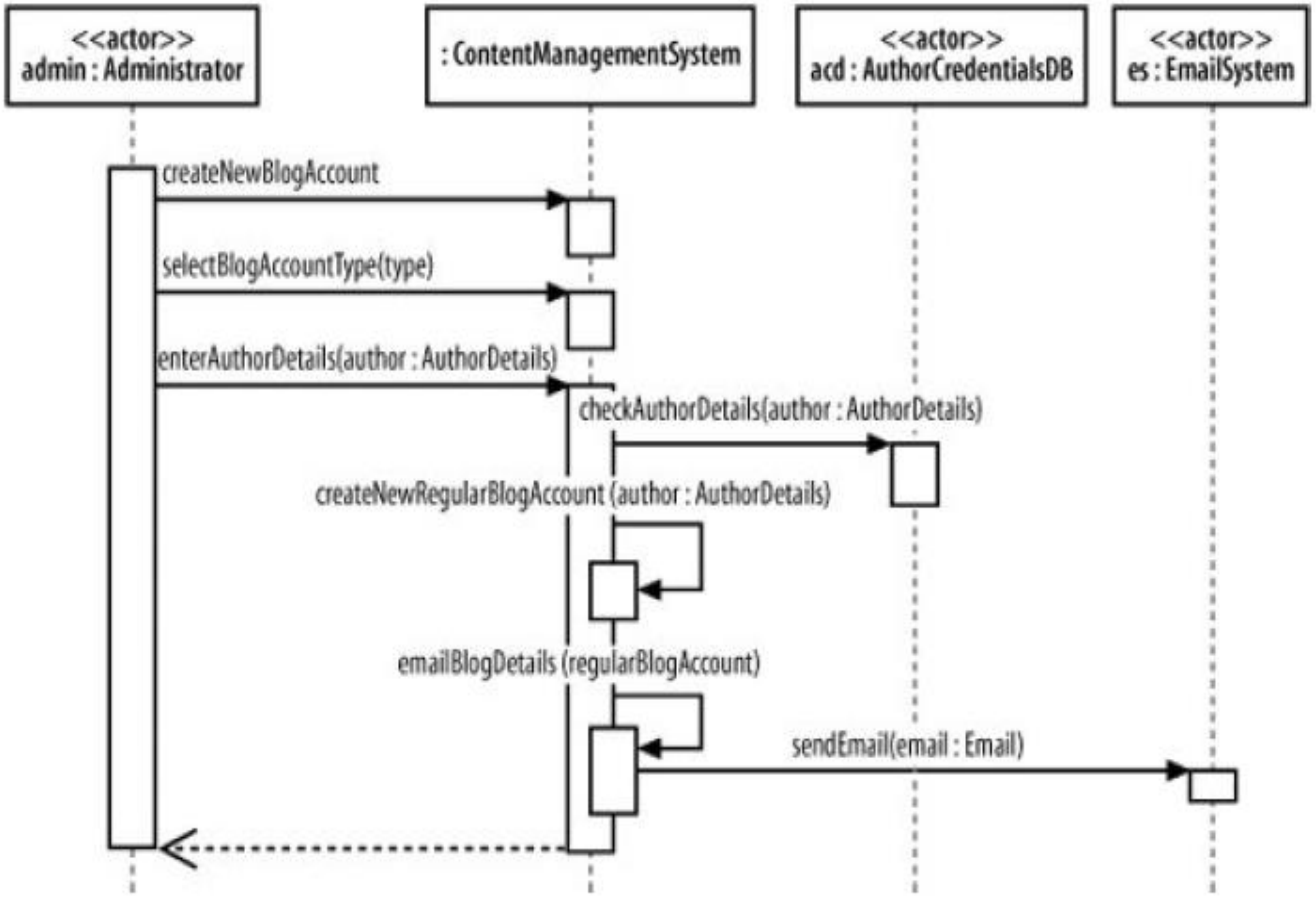
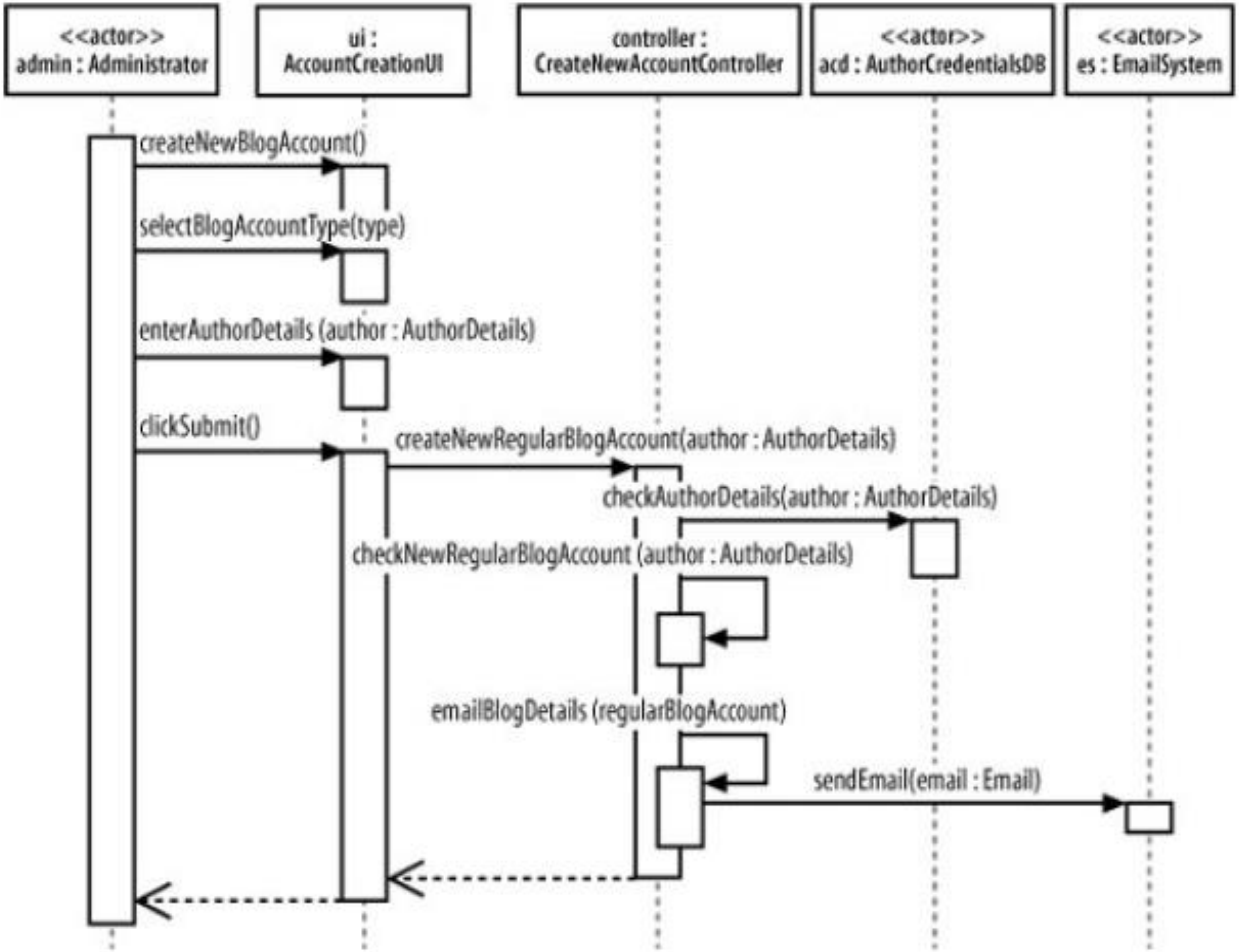

<<actor>> admin : Administrator
: ContentManagementSystem
<<actor>> acd : AuthorCredentialsDB
<<actor>> es : EmailSystem
createNewBlogAccount
selectBlogAccountType(type)
enterAuthorDetails(author : AuthorDetails)
checkAuthorDetails(author : AuthorDetails)
createNewRegularBlogAccount (author : AuthorDetails)
emailBlogDetails (regularBlogAccount)
sendEmail(email : Email)

Figure 7-15. Adding more detail about the internals of your system


<<actor>>
admin : Administrator
ui :
AccountCreationUI
controller :
CreateNewAccountController
<<actor>>
acd : AuthorCredentialsDB
<<actor>>
es : EmailSystem
createNewBlogAccount()
selectBlogAccountType(type)
enterAuthorDetails (author : AuthorDetails)
clickSubmit()
createNewRegularBlogAccount(author : AuthorDetails)
checkAuthorDetails(author : AuthorDetails)
checkNewRegularBlogAccount (author : AuthorDetails)
emailBlogDetails (regularBlogAccount)
sendEmail(email : Email)

Semantics

O1	Object O1	Ali
:C	an anonymous object from classe C	:PERSON
/R:C	an anonymous object from classe C playing the role R	/Reader:PERSON
/R	an anonymous object playing the role R	/Reader
O/R:C	An object O from classe C playing the role R	Ali/Reader:PERSON

Managing Complex Interactions with **Sequence Fragments**

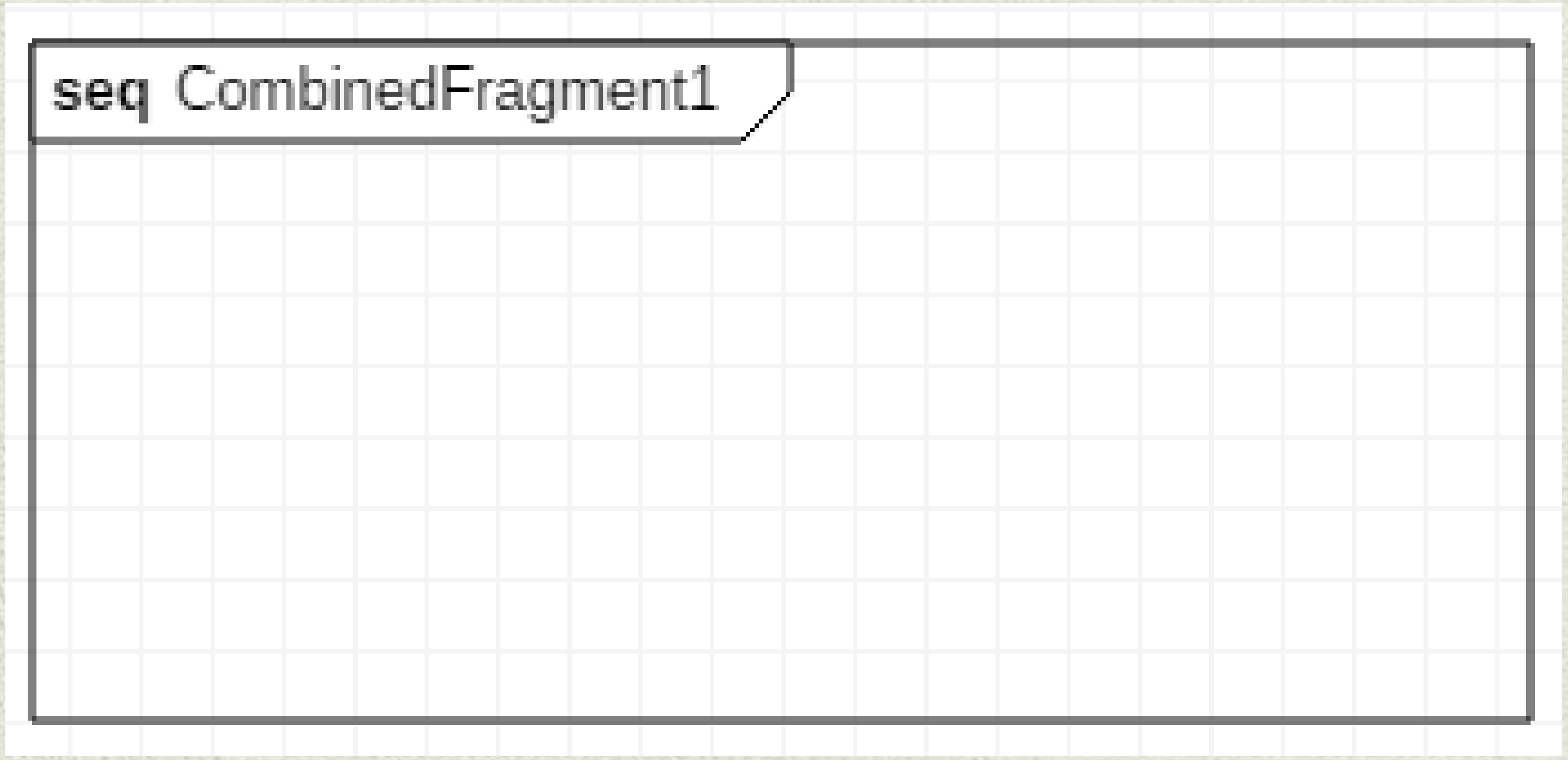
- **When to use** sequence fragments.
- **What is a sequence fragment** in sequence diagrams.
- **Types of sequence fragments.**
- **Examples** of using sequence fragments in a sequence diagram.

When to use Sequence fragments ?

- The focus of our last session was on **understanding the purpose of sequence diagram** as well as using **the basic notations for drawing simple interactive sequence diagram.**
- However, those notations we have used so far, are not sufficient if we want **to show complex interactions such as loops and alternate flows in a diagram.**

What is a Sequence fragment ?

- A sequence fragment (*Interaction fragment* or *combined fragment*) is represented as a box that encloses a portion of the interactions within a sequence diagram.



What is a Sequence fragment ?

Typically, a sequence fragment consists of :

- Fragment's box
- Interactions/Messages
- Operator
- One or more interaction operand
- Guard condition/s

How does it look like ?

Example of a sequence fragment-alt

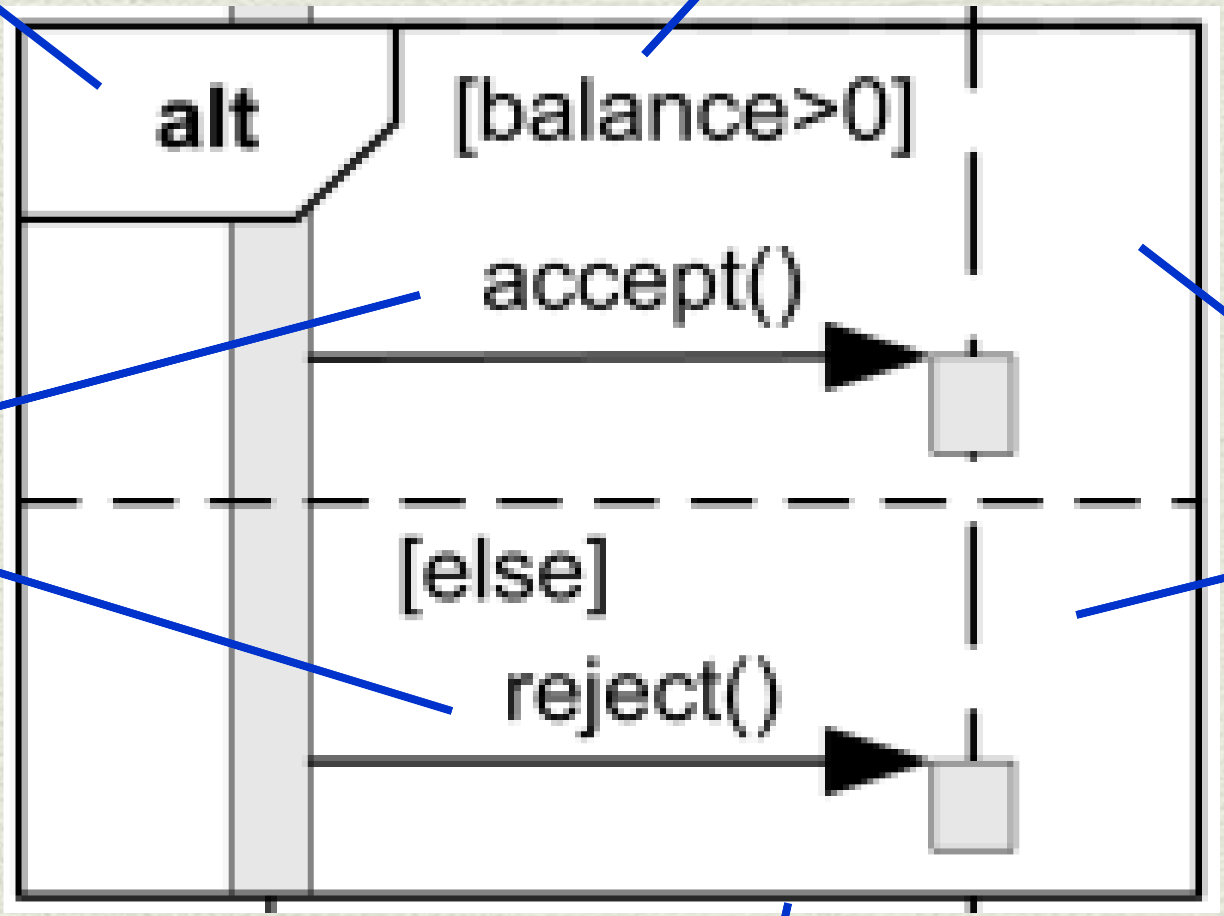
sequence fragment's type **operator**

Guard condition a parameter that is tested on entry to the fragment

messages /interactions

Interaction operands

sequence fragment's box

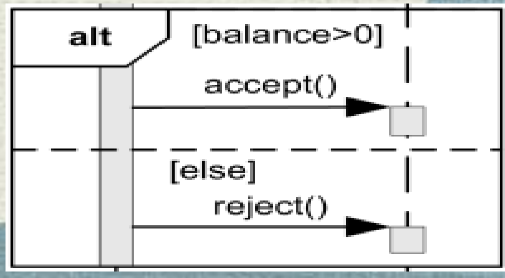


What is a Sequence fragment ?

A sequence **fragment's box** overlaps the region of the sequence diagram where the fragment's interactions take place.

A fragment box can contain any number of **interactions** and, for large complex interactions, further **nested fragments** as well.

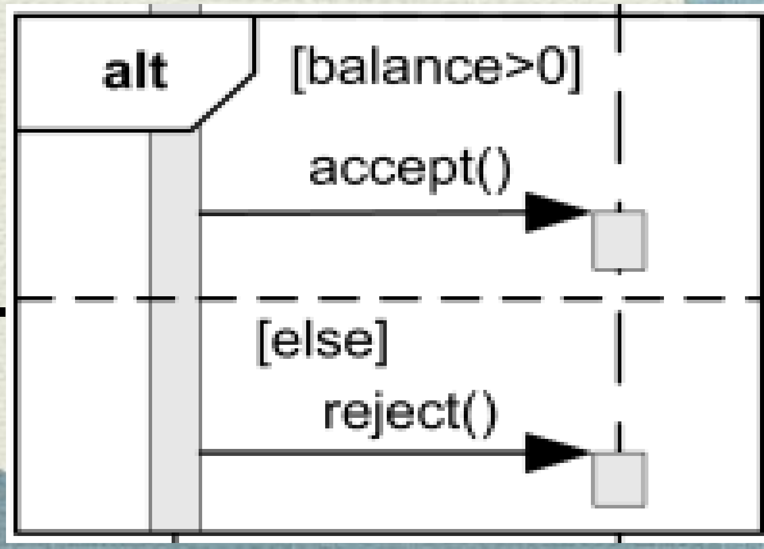
The top left corner of the fragment box contains an **operator**. The fragment operator indicates **which type of fragment this is.**



What is a Sequence fragment ?

- Every sequence fragment contains at least one **interaction operand**, which can contain messages and smaller combined fragments.
- Use **Guard to describe the conditions** in which the messages inside it are performed.
- For example, in a **Loop** combined fragment, you can use the guard to specify the condition during which the loop continues.
- In an **alt** combined fragment, you can specify

  a separate condition for each interaction operand.



Types of Sequence

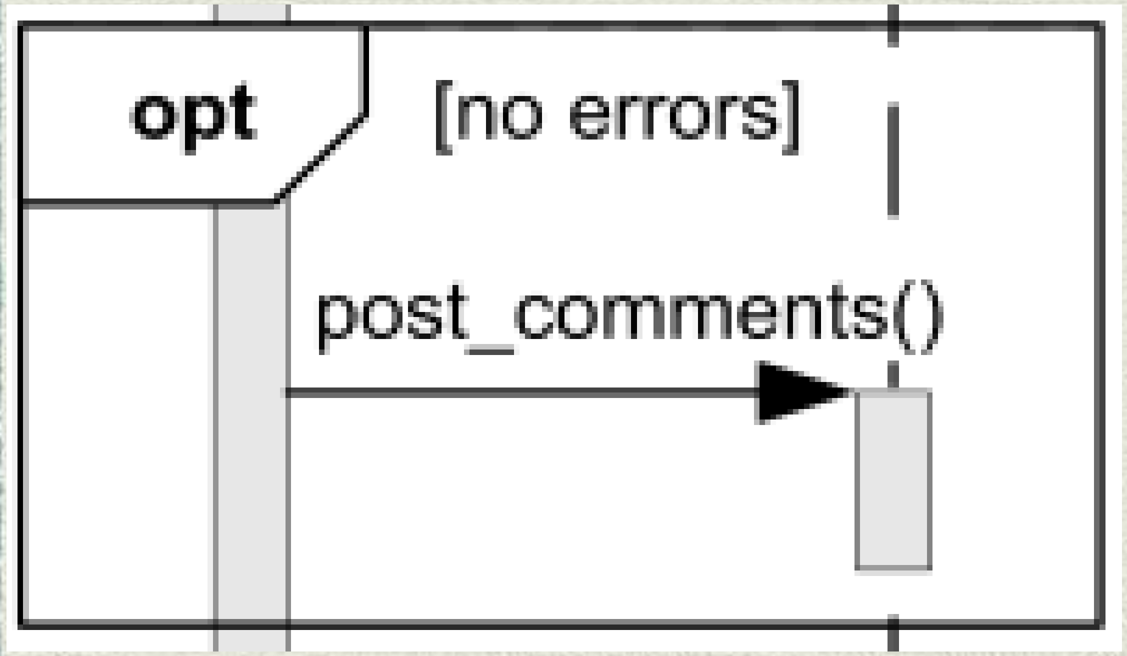
There are several kinds of sequence fragments, next slides present five of sequence fragment and explanations .

Examples of sequence fragments:

- **Option - opt**
- **Alternatives- alt**
- **Iteration - loop**
- **Break – break**
- **Parallel - par**

Option - opt

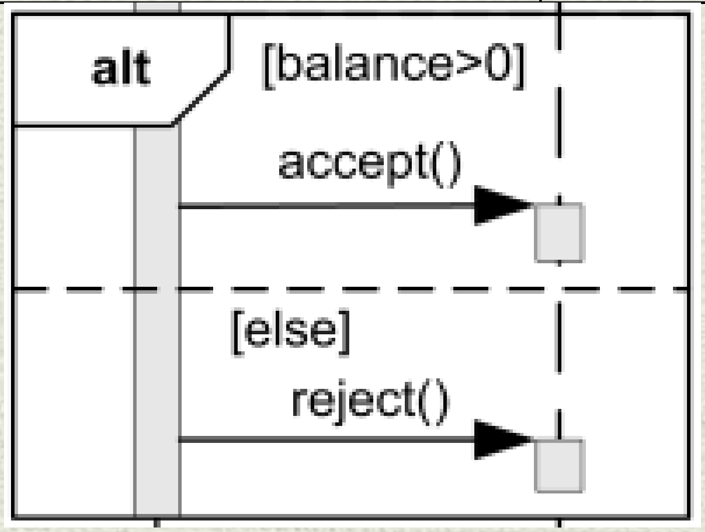
Type	Parameters	Why is it useful?
opt	**[guard_condition]**	▪The interactions contained within this fragment will execute only if the guard condition evaluates to true. ▪Similar to **a simple if(..) statement** in code with no corresponding else.



Post comments if there were no errors.

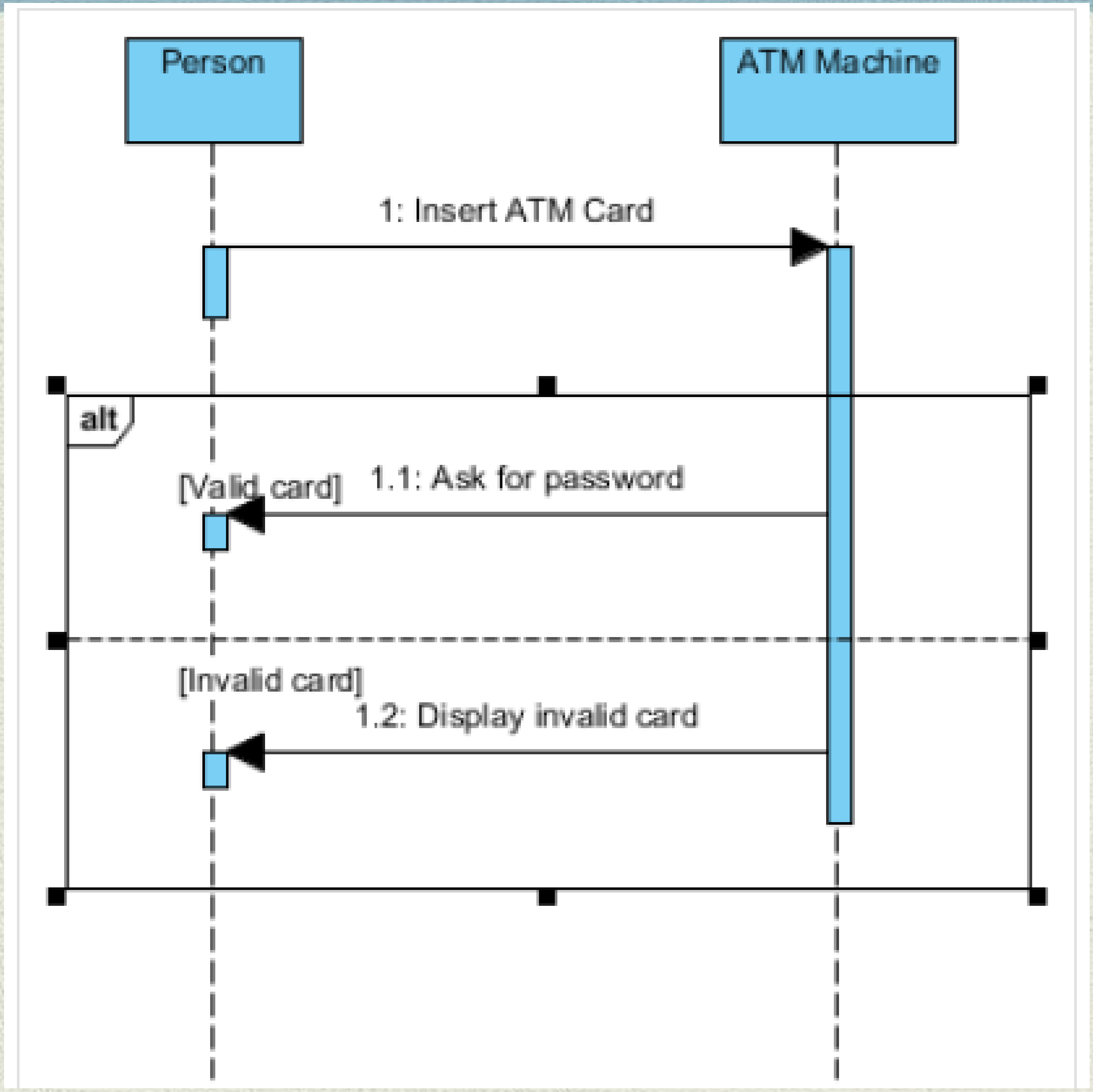
Alternatives- alt

Type	Parameters	Why is it useful?
alt	**[guard_condition1]** **…** **[guard_condition2]** **…** **[else]**	▪ Helps you specify that a set of interactions will be executed only under certain conditions. ▪ Similar to an **if(..) else** statement in code. Depending on which guard condition evaluates to true first, the corresponding sub-collection of interactions will be executed.



call `accept()` if `balance > 0`, call `reject()` otherwise.

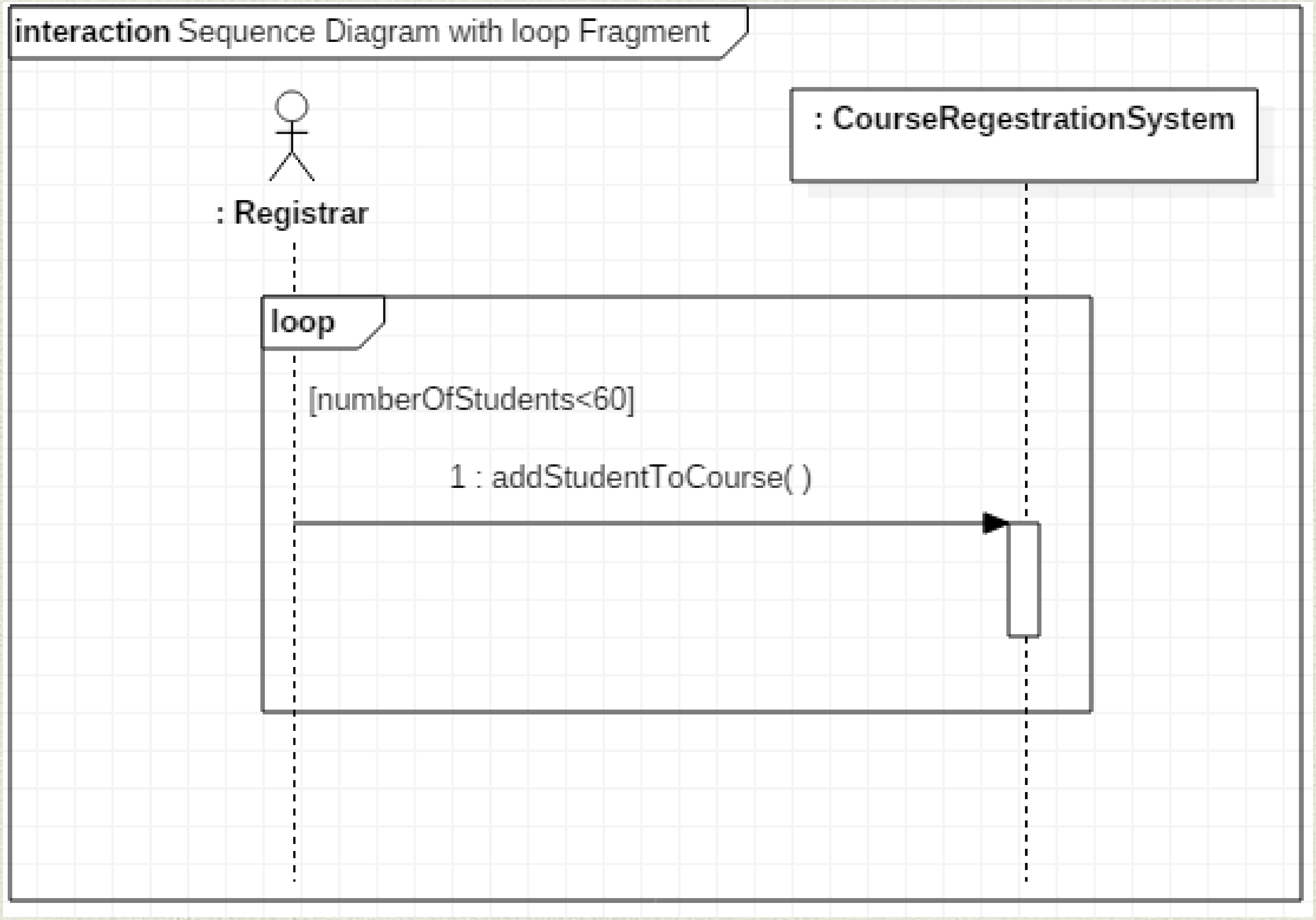
Alternatives- alt example



Iteration - loop

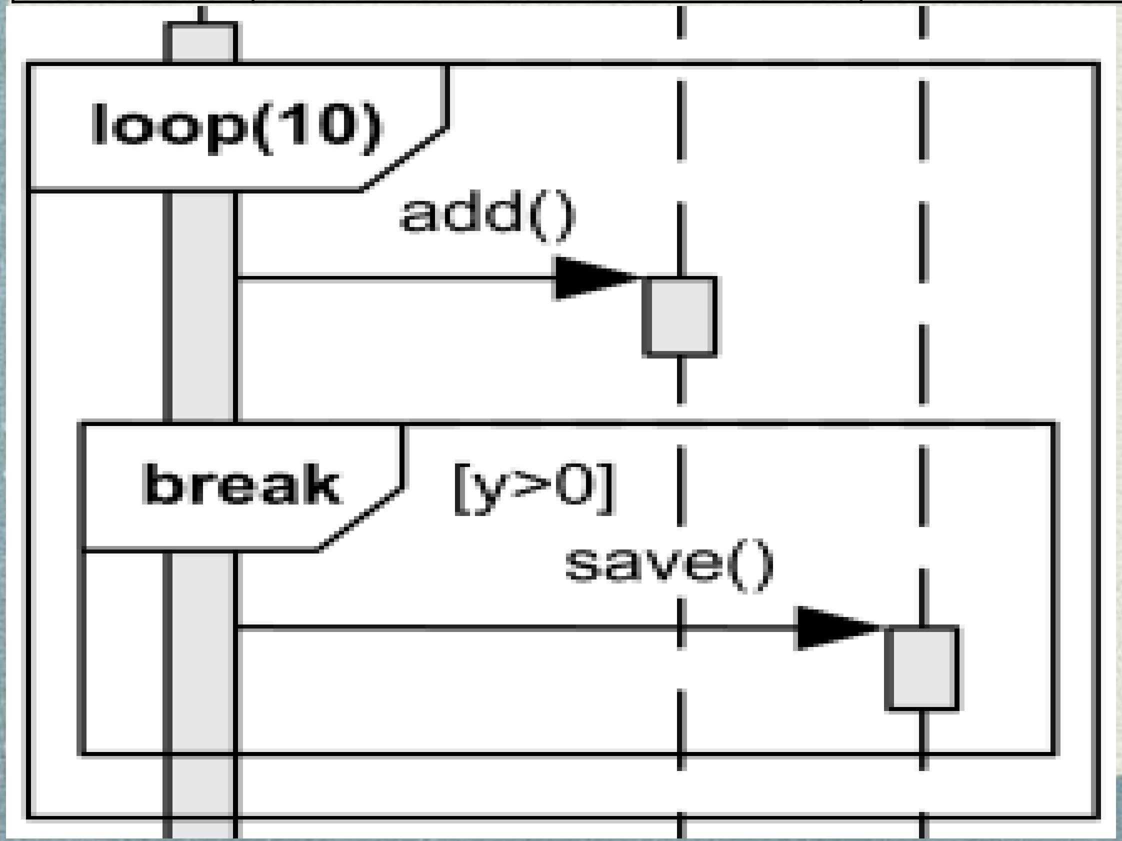
Type	Parameters	Why is it useful?
loop	**min times,** **max times,** **[guard_condition]**	▪Loops through the interactions contained within the fragment a specified number of times until the guard condition is evaluated to false. ▪Very similar to the Java and C# **for(..) loop**. ▪ **Useful when you are trying execute a set of interactions a specific number of times** ▪Loop combined fragments have the properties **Min** and **Max, which indicate the minimum and maximum number of times that the fragment can be repeated. The default is no restriction.**

Iteration – loop example


interaction Sequence Diagram with loop Fragment
: Registrar
: CourseRegestrationSystem
loop
[numberOfStudents<60]
1 : addStudentToCourse( )

Break – break

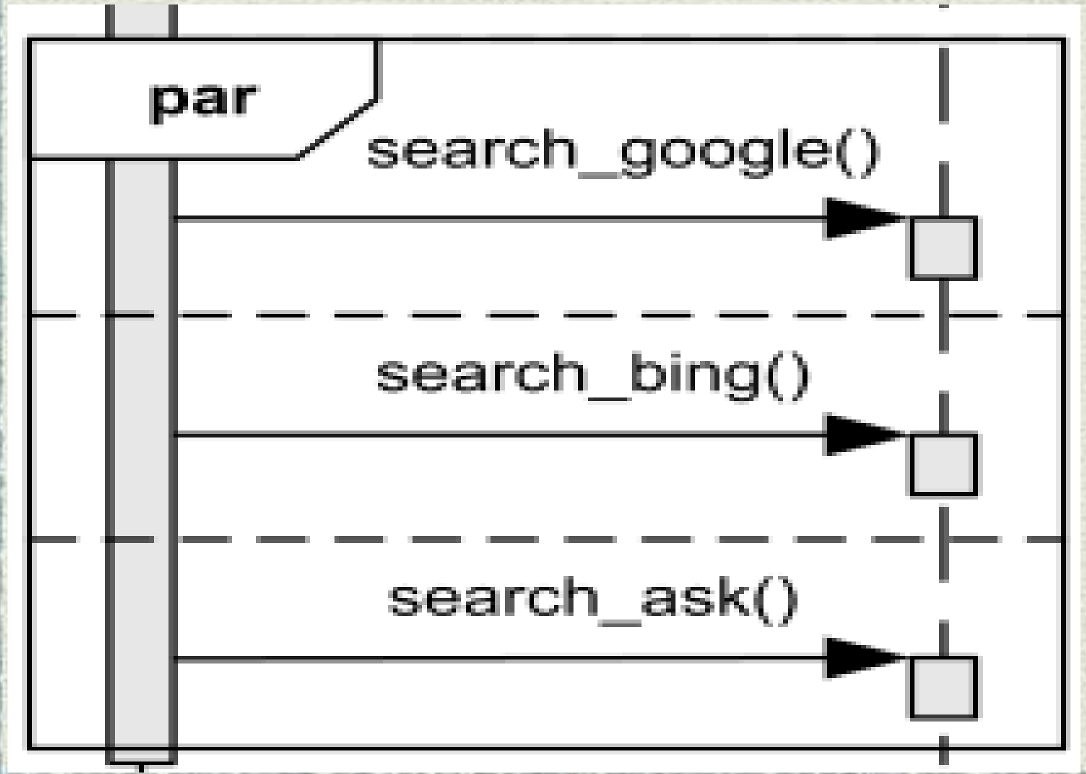
Type	Parameters	Why is it useful?
break	[guard_condition]	▪If this fragment is executed, the rest of the sequence is abandoned. ▪You can use the guard to indicate the condition in which the break will occur. ▪the break combined fragment is much like the break keyword in a programming language like C++ or Java. ▪Breaks are most commonly used **to model exception handling.**



Break enclosing loop if `y>0`.

Parallel - par

Type	Parameters	Why is it useful?
par	**None**	▪Helps you to show parallel processing activities in a sequence diagram . ▪The frame's content section can be broken into horizontal operands separated by a dashed line. Each operand in the frame represents a thread of execution done in parallel.



Search Google, Bing and Ask in parallel.

References

- Fowler, M. (2004). UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional.
- Miles, R and Hamilton, K. (2006) Learning UML 2.0. Sebastopol: O'Reilly Media, Inc.
- Pender, T (2003). UML Bible. John Wiley & Sons, Inc., New York, NY.
- Pilone, D., & Pitman, N. (2006). *UML 2.0 in a nutshell*. O'Reilly.