Introduction to Sequence Diagrams

Davide Falessi

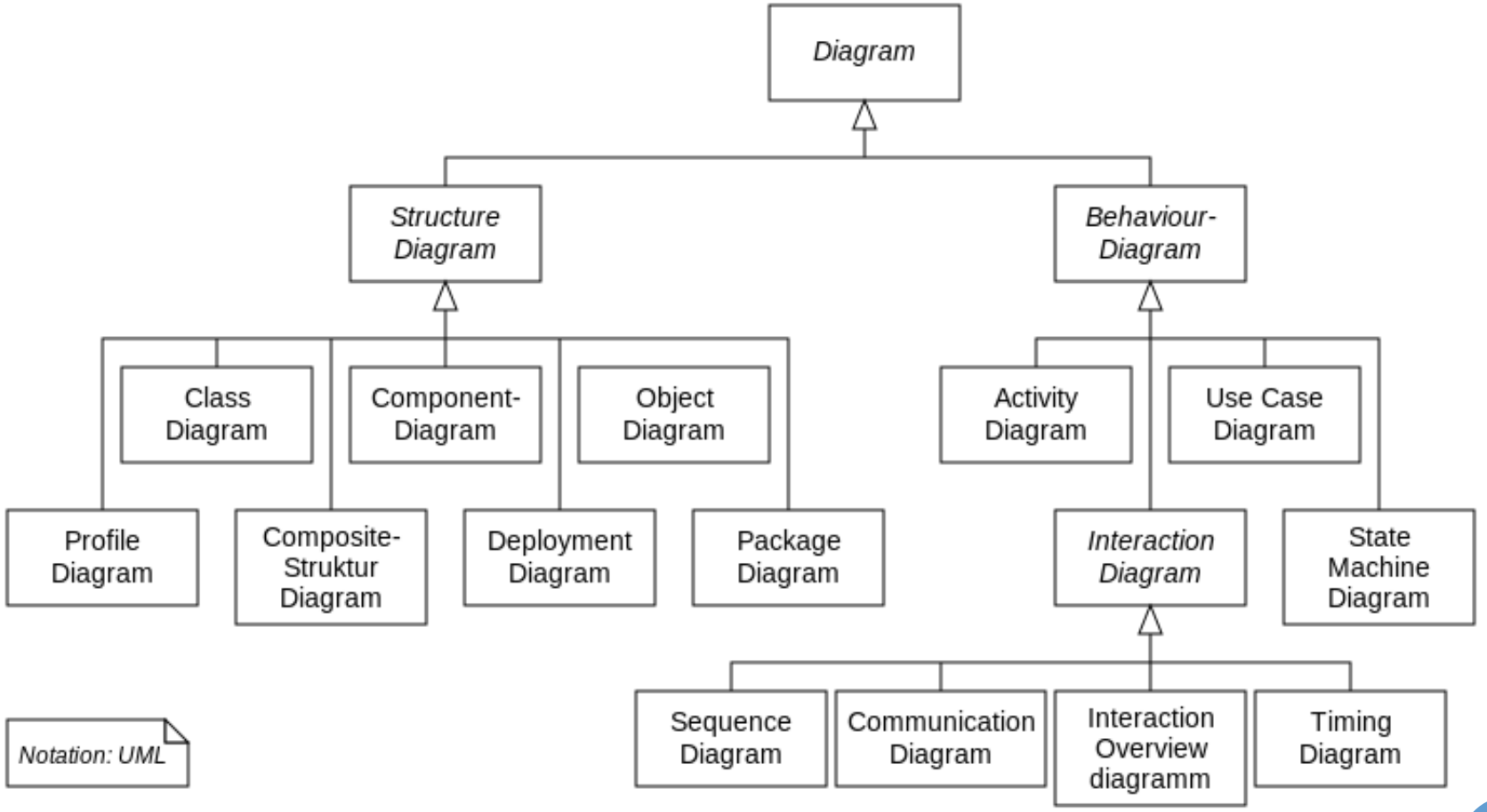
Sequence Diagram

- Title: Learning UML 2.0
- By: Russ Miles, Kim Hamilton
- Publisher: O'Reilly Media
- Assumption: Class diagram already done in previous courses, pages 63-100
- Today: pages 108-130

Sequence diagrams

- Sequence diagrams are an important member of the group known as **interaction** diagrams.
- Interaction diagrams model important **runtime** interactions between the **parts** that make up your system and form part of the logical view of your model

UML DIAGRAMS



Scope

- 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** and in what **order** those interactions will occur.

Layout

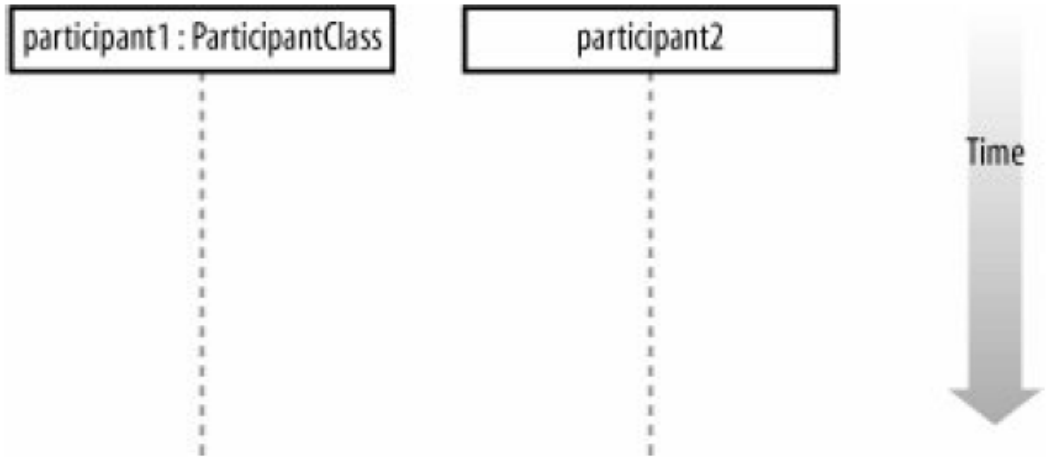
- A sequence diagram is made up of a collection of **participants**: the parts of your system that interact with each other during the sequence.
- **Where** (vertically) a participant is placed on a sequence diagram is important.
- Regardless of where a participant is placed vertically, participants are always arranged **horizontally** with no two participants **overlapping** each other.
- Each column is an object.

Time

- 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 Example

Figure 7-3. Time runs down the page on a sequence diagram in keeping with the participant lifeline

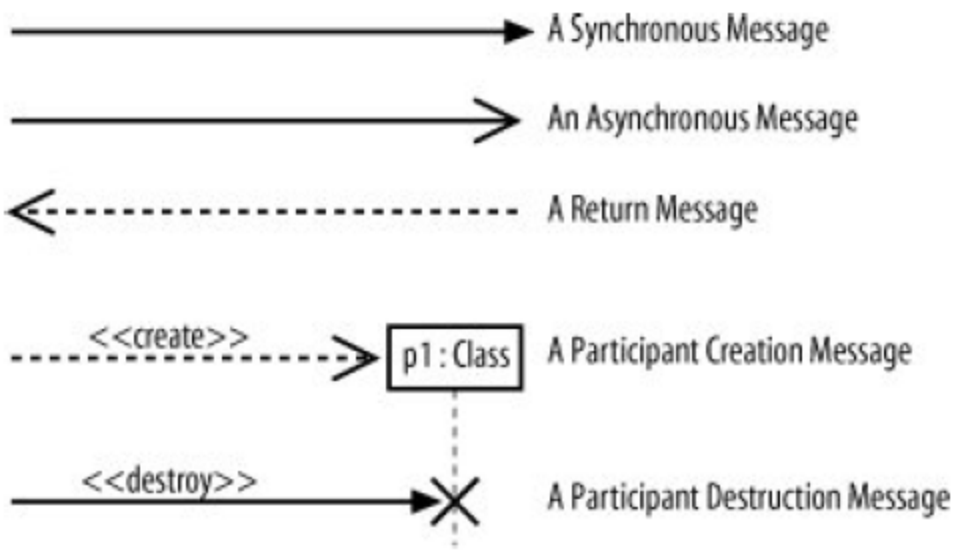


Time, duration and order

- **Time on a sequence diagram is all about ordering, not duration.**
- Although the time at which an interaction occurs is indicated on a sequence diagram by where it is placed vertically on the diagram, how much of the vertical space the interaction takes up has nothing to do with the duration of time that the interaction will take. (i.e., **proportions are not important** at all)

Types of messages

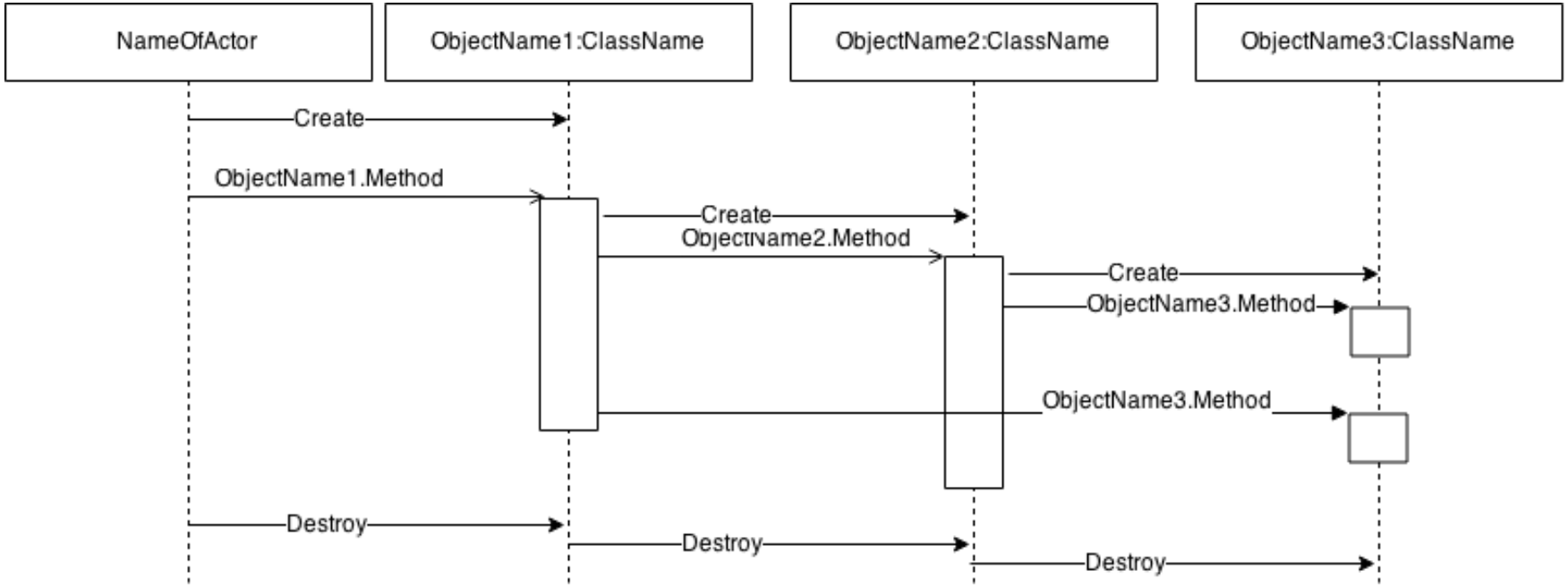
Figure 7-8. There are five main types of message arrow for use on sequence diagram, and each has its own meaning



Sometimes implicit and on the dash line

The dash line continues

Sequence Diagram in OO



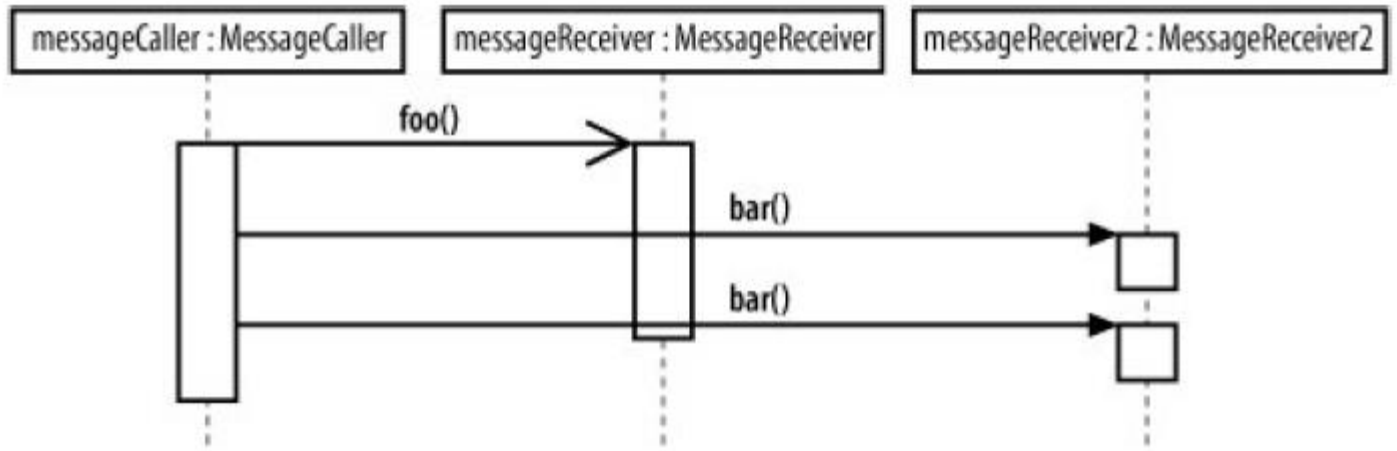
Asynchronous messages

- Interactions can happen at the same point in time, and sometimes you will want to initiate a collection of interactions all at the same time and not wait for them to return at all.
- Example: *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.*

ASYNCHRONOUS = NOT WAITING

- 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.

Figure 7-10. While the foo( ) message is being worked on by the messageReceiver object, the messageCaller object has carried on with the interaction by executing further synchronous messages on another object

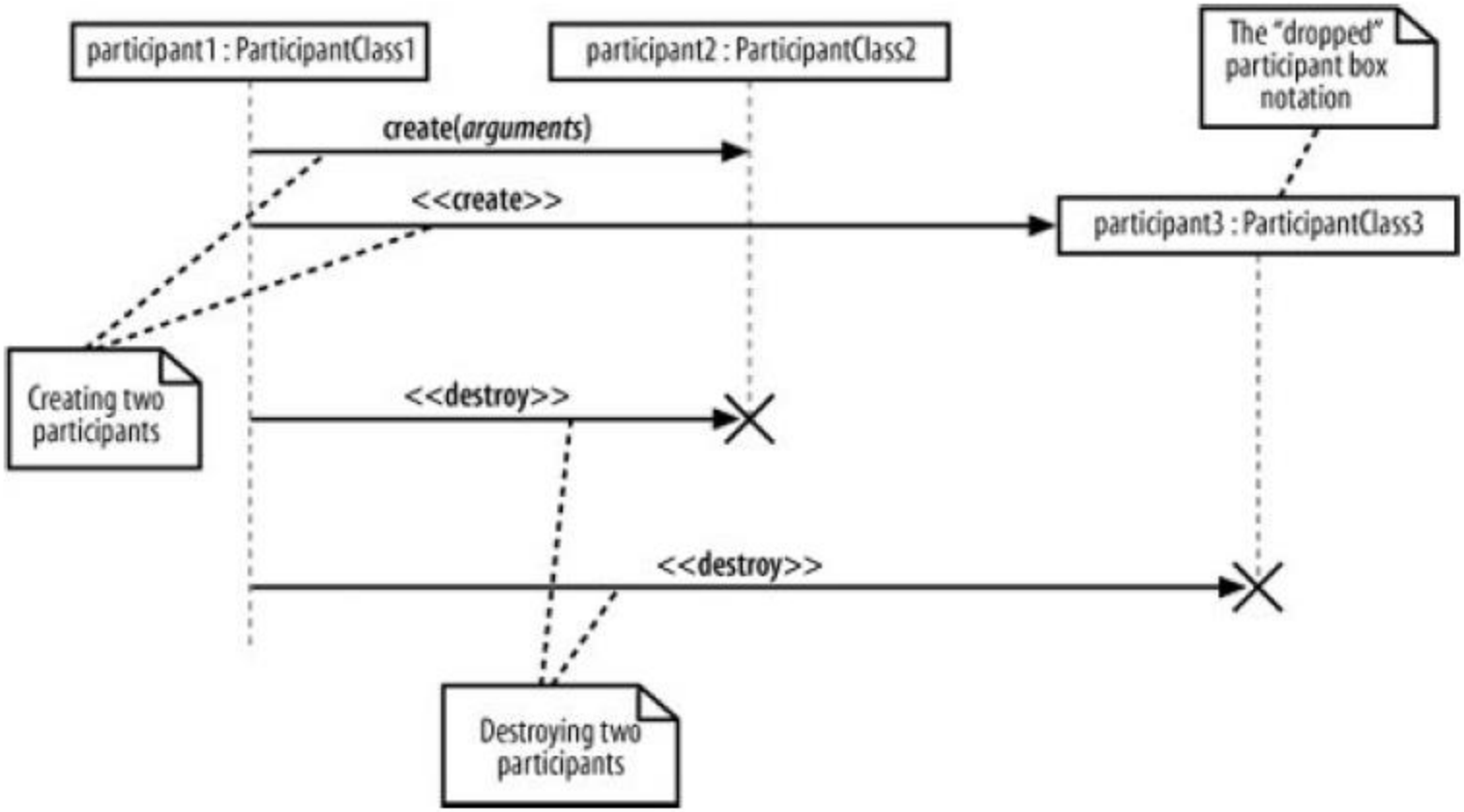


Return message

- It is optional.
- In general, it is used to highlight that /when something is provided back to the original caller rather than just a “change” in the receiver.

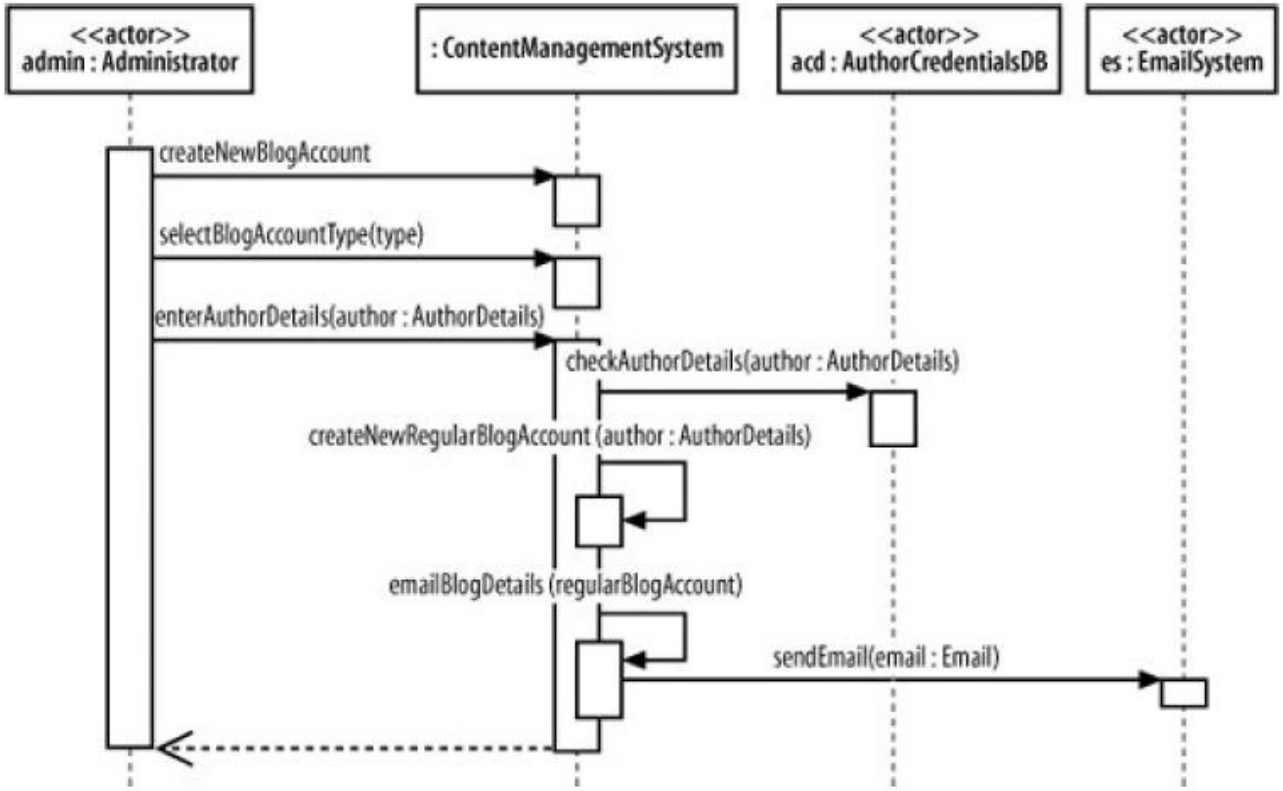
Creation and destruction

Figure 7-11. Both participant2 and participant3 are created throughout the course of this sequence diagram



Actors

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



Common Mistakes

- Having an outgoing arrow from a line (i.e., not a box) if not an actor and vice versa.
- Starting a box from a place different than an incoming arrow.
- Having two arrows incoming to the same box if the first arrow is synchronous. If it is asynchronous it is odd but valid.
- Having a box that is shorter than the box created by its synchronous outgoing arrow.