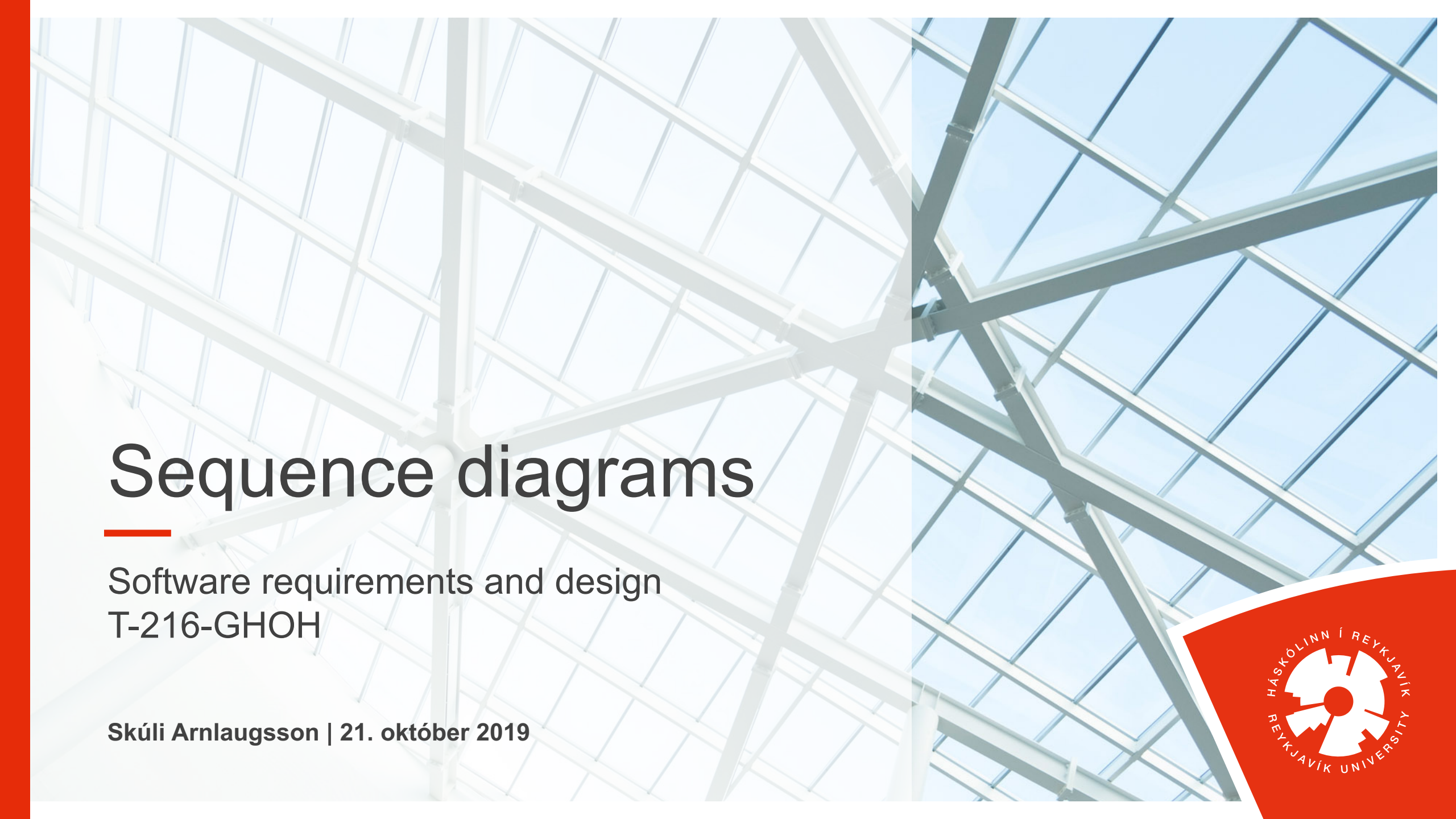


Sequence diagrams

Software requirements and design
T-216-GHOH

Skúli Arnlaugsson | 21. október 2019

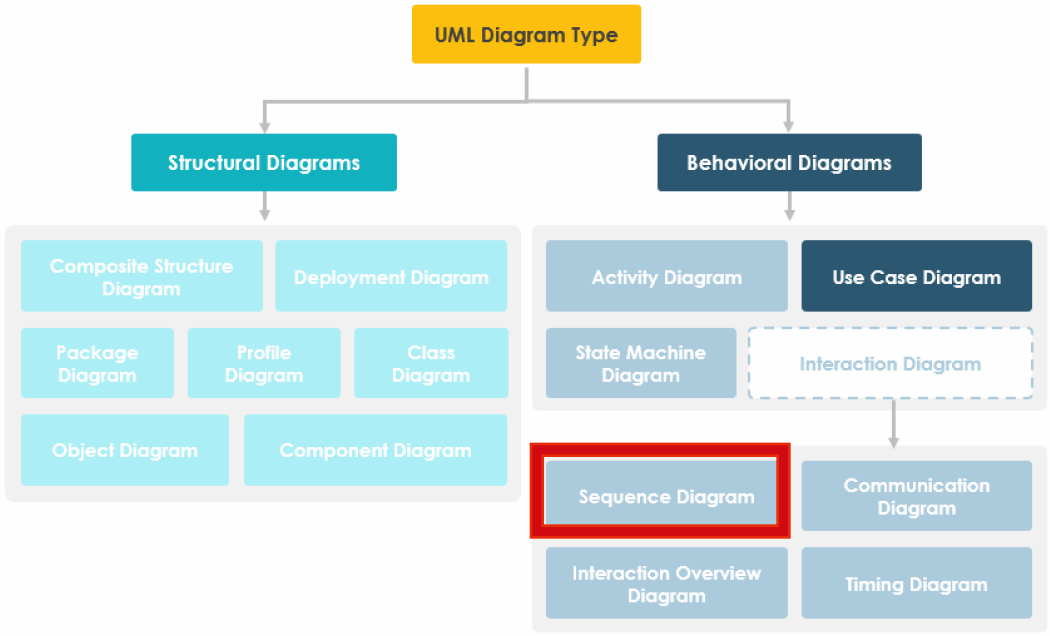
This lecture

- Purpose of sequence diagrams
- Sequence diagram notation
  - Basics
  - Object lifecycles
  - Different types of calls
  - Interaction frames

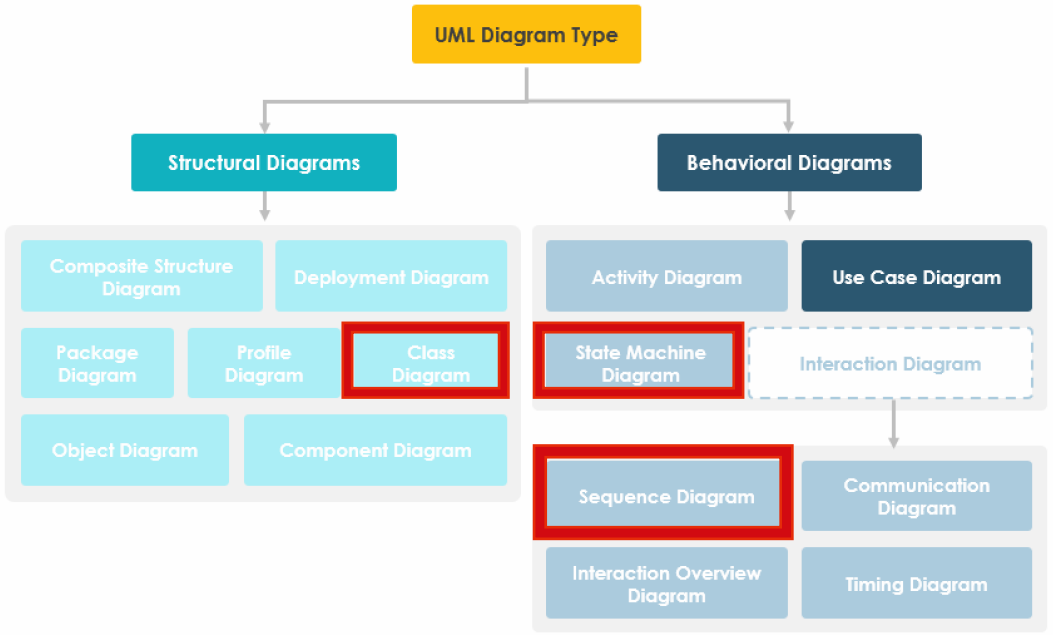
Purpose of Sequence Diagrams

Well, what is a sequence diagram?

- Sequence diagrams are interaction diagrams (*ísl. samskiptarit*).
- Their purpose is to model **how objects interact with each other in a single use case** (*ísl. notkunartilvik*).
- As with all the diagrams we have encountered, sequence diagrams can be drawn to shown different levels of abstraction and different levels of details.

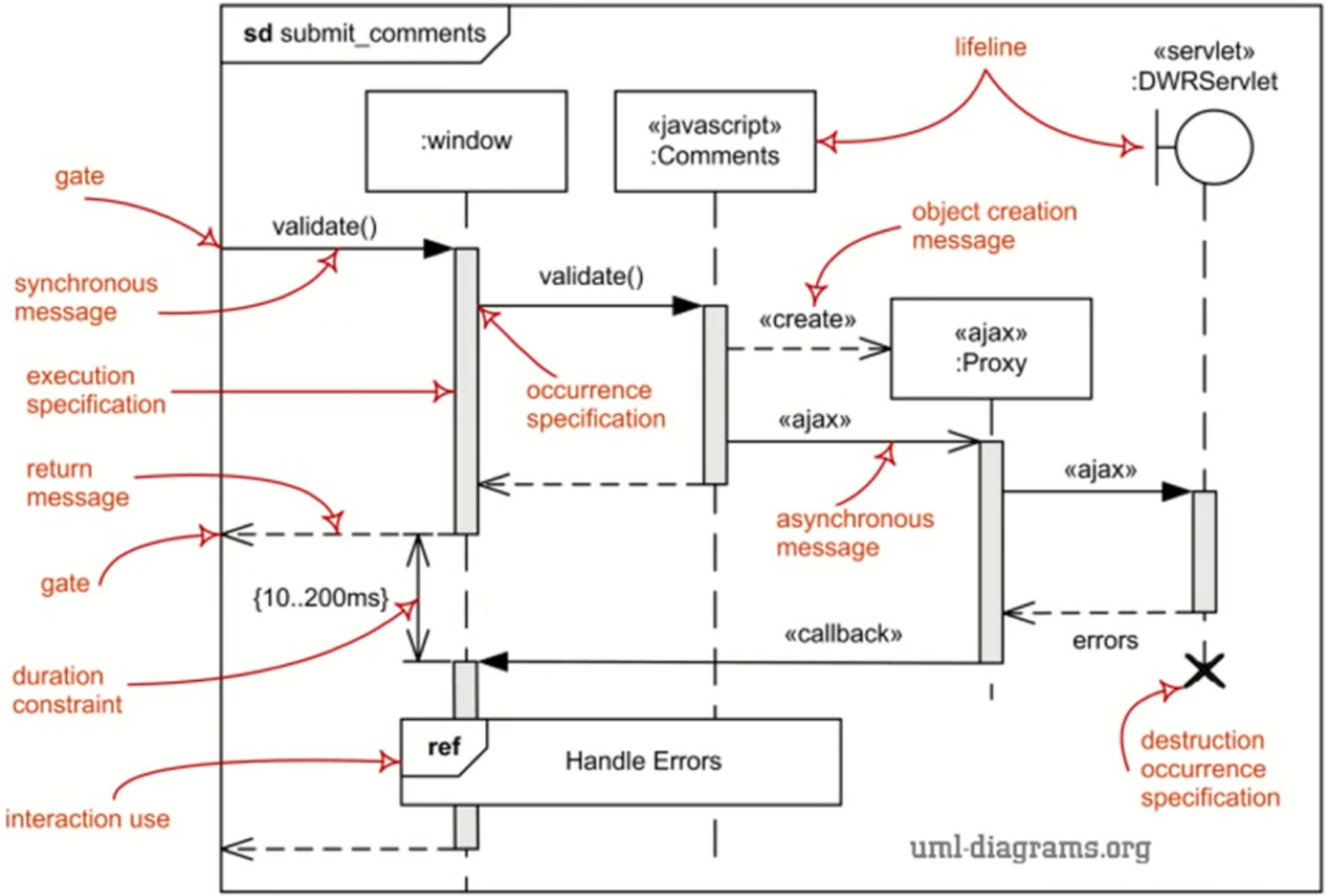


Big picture: Our UML Diagrams



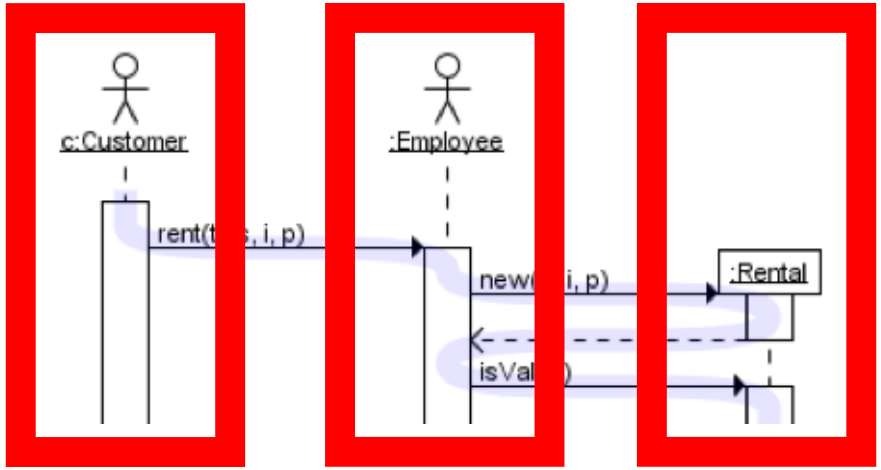
- The class diagram (ísl. klasarit) details our system's main objects, and how they are connected. The structure of our system.
- The state machine diagram (ísl. stöðurit) details the how the system or parts of the systems behave across multiple use cases.
- The sequence diagram shows a single use case and how objects (from our class diagram) interact with each other during the use case.

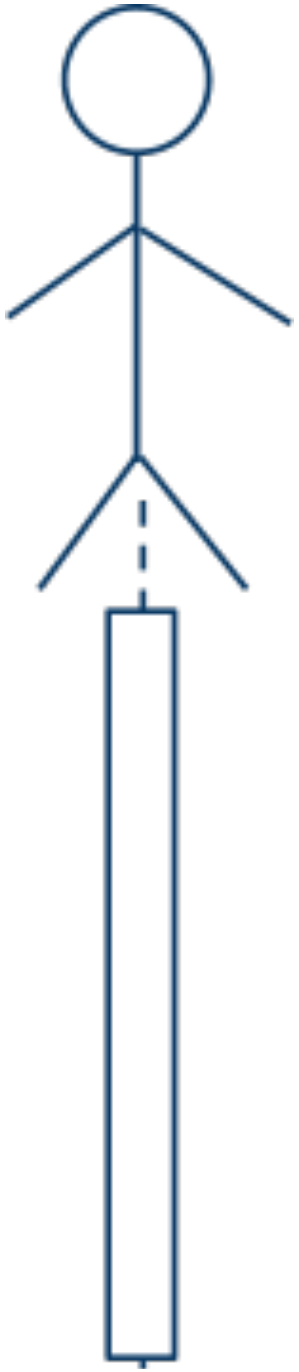
Sequence diagram basic notation


sd submit_comments
lifeline
«servlet»
:DWRServlet
:window
«javascript»
:Comments
gate
validate()
object creation
message
synchronous
message
validate()
«create»
«ajax»
:Proxy
execution
specification
occurrence
specification
«ajax»
return
message
«ajax»
asynchronous
message
gate
{10..200ms}
«callback»
errors
duration
constraint
ref
Handle Errors
destruction
occurrence
specification
interaction use
uml-diagrams.org

The basics

- Sequence diagrams are read like from left to right, top to bottom.
- A sequence diagram will show at least two object types / instances.
- In most cases each type of object or instance will be either an actor, shown as a stick figure or a class, denoted in a rectangle with the class name.



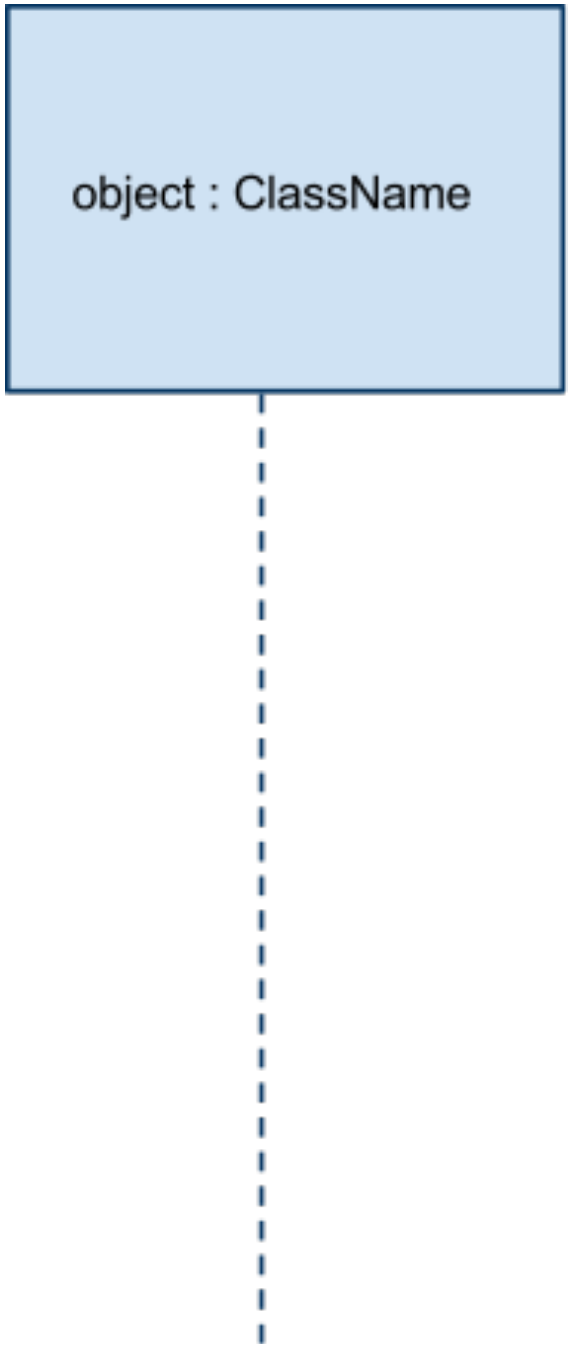


Actors

- For most use cases, they are initiated with an actor starting it. That actor is then placed to the far left as a stick figure.
- More actors can be involved in the use case, but rectangles can also be used for classes (or their instances).

The lifeline

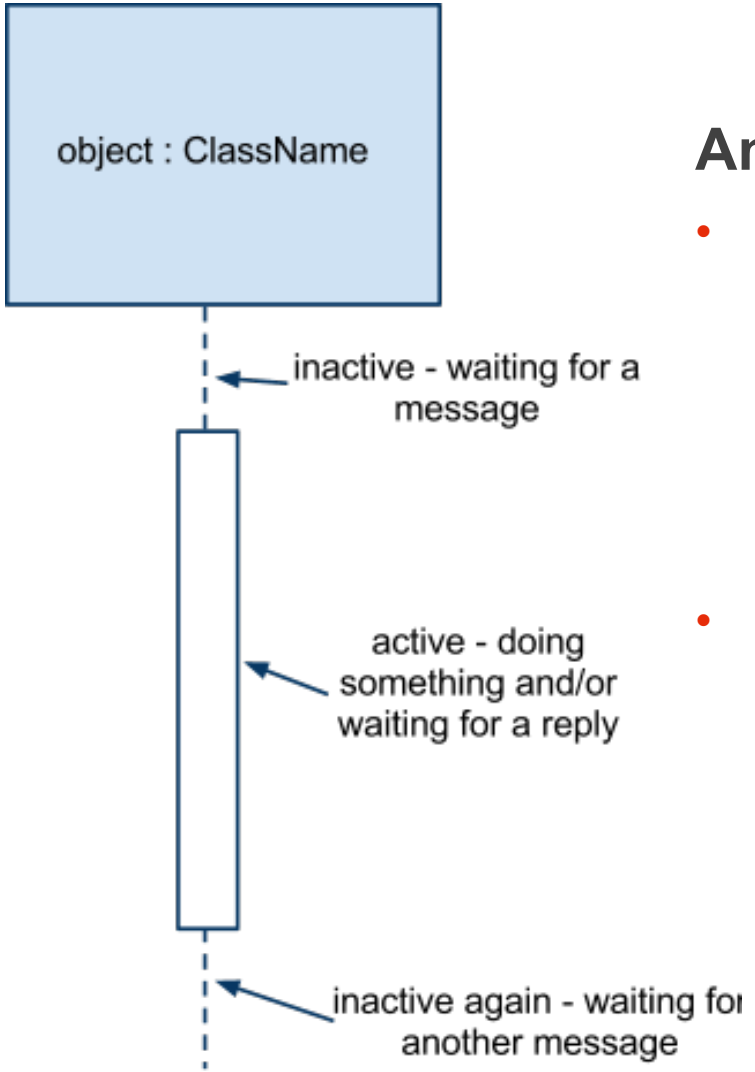
- The lifeline is shown as a dashed line, indicating that the instance is ”alive” (ready, available, awake, or present).
- The lifeline goes down from the actor or class box. It may run all the way to the bottom of the diagram, or it may be terminated earlier, as we’ll see in a bit.



Active objects

An object can be active

- In an active state, the instance is doing something, or waiting for something to be completed elsewhere.
- Being in an active state most of the time prevents the instance from doing anything else until the task is completed.
- An instance being active is indicated with a rectangle on top of the instance's lifeline.



An object can be inactive

- Being in an inactive state simply indicates that the instance isn't doing something in the use case, nor is it waiting for someone else.
- It can be taken from an inactive state if something else requests something from it.

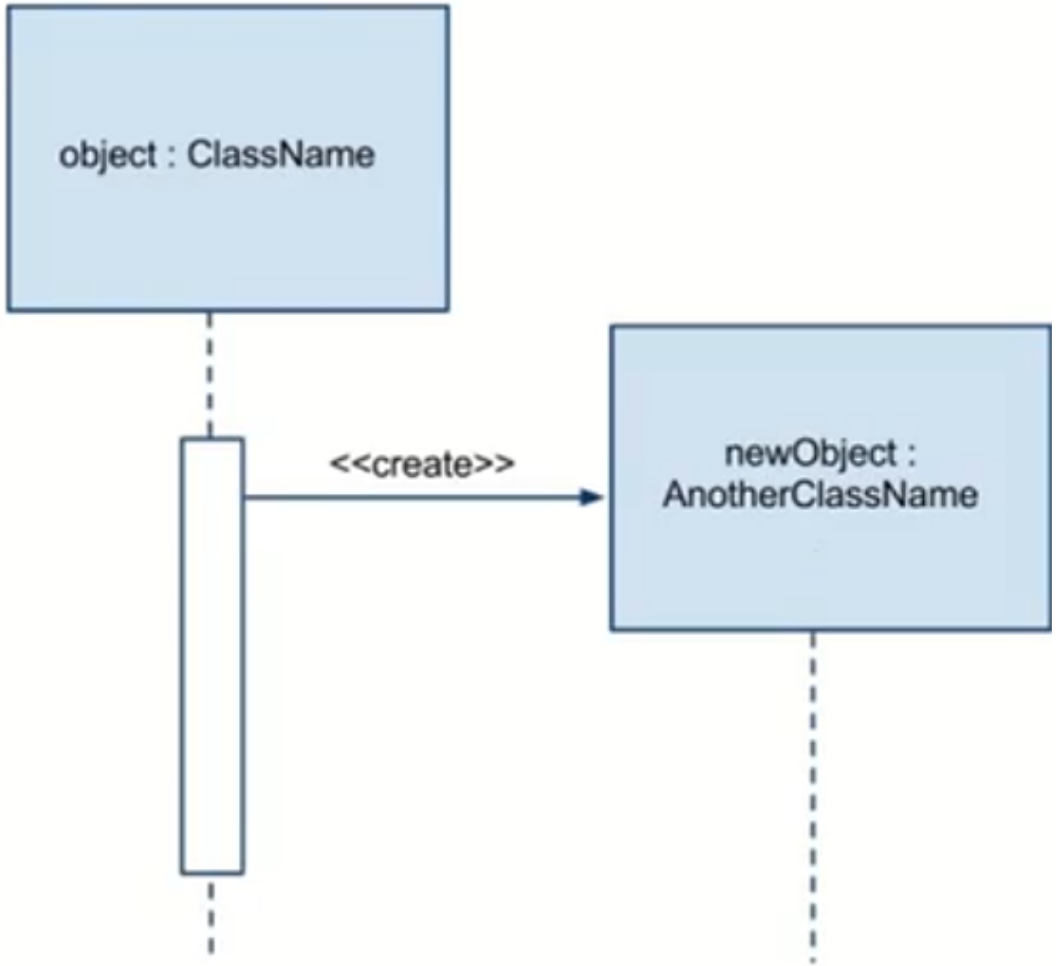
Object lifecycles

Object instance creation

- An instance can be created during an use case. In sequence diagrams this is denoted in a special way.
- An instance is usually created at the request of another object's instance, and the message that gets sent is normally named:

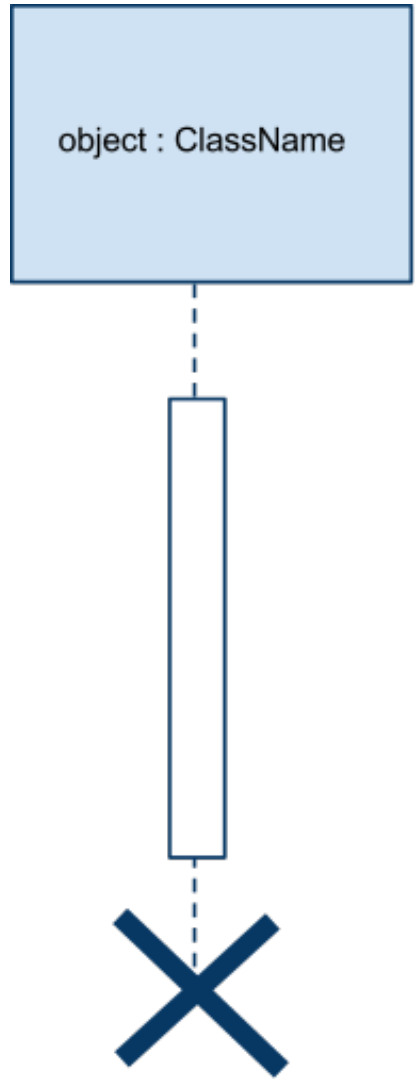
```
<<create>>
```

- The rectangle and lifeline for that instance are shown when they are created (not at the top alongside other instances and actors).



Object instance deletion

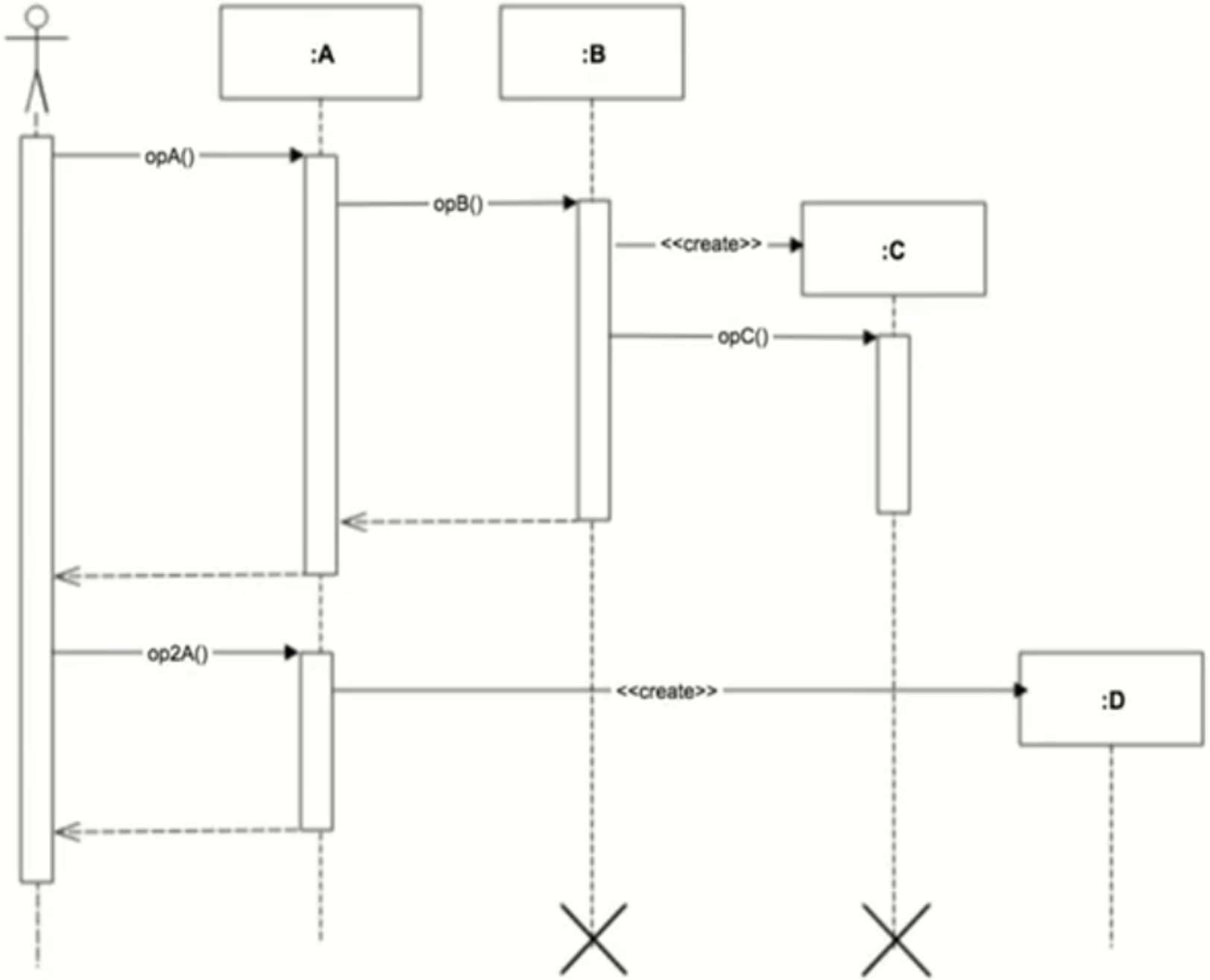
- When an instance is deleted for various reasons, the lifeline is ended for the instance and black cross (X) is put at the end of the lifeline.
- There could be various reasons why this is done. Most obvious one is that the instance is deleted during the use case.



Object instances lifetimes

The lifetime of object instances can be categorized into 4 groups:

- An object exists the whole use case – it existed before the use case started, and it will exist after the use case is completed.
  - Example: Library member during the use case for taking a book at the library.
- An object is created during the use case and will exist when it's created.
  - Example: New Library member during the use case for applying for a library card.
- An object exists at the start of the use case but is deleted during it.
  - Example: A bank account is closed during the use case.
- An object is both created and deleted during the use case.
  - Example: An personal hotspot connection via a mobile phone is started and turned off during the use case.



Object lifetimes

- Instance of class A exist throughout the use case.
- Instance of class B is deleted during the use case, but existed at the start of it.
- Instance of class C was created and created during the use case.
- Instance of class D is created during the use case and still exists at the end of it.

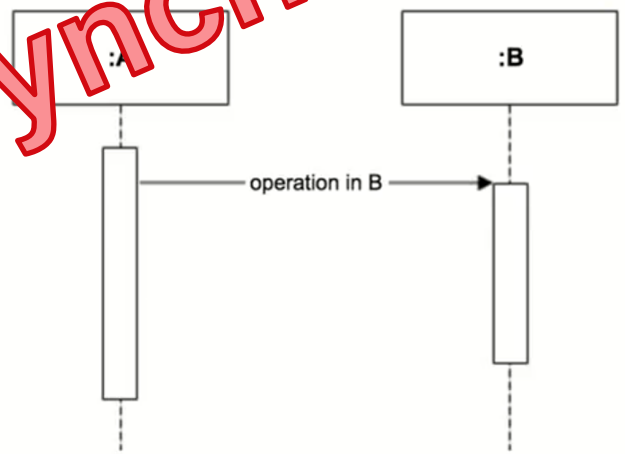
Different types of calls

Operations and messages

Operation

When an object “calls” a method/function/operation in another object, and waits for the operation to be completed, it is represented with a filled arrowhead.

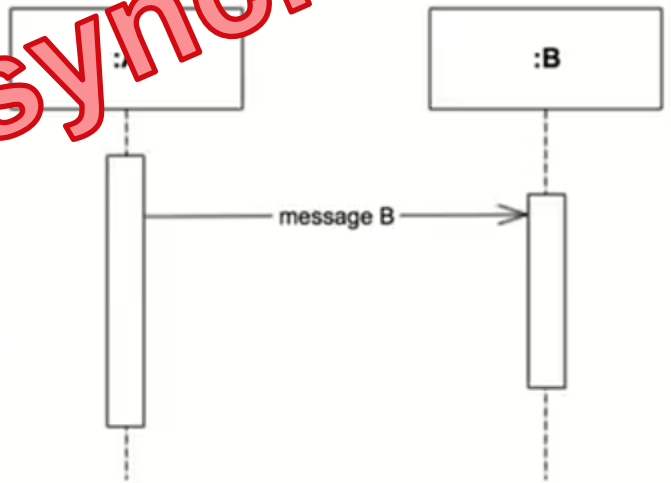
Synchronous



Message

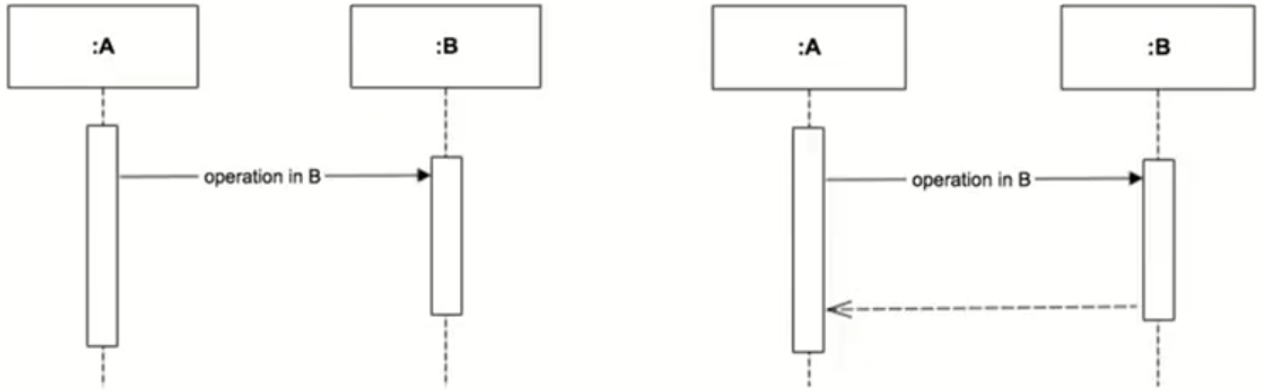
When an object “calls” a method/function/operation in another object, and doesn’t wait for the operation to be completed, it is represented with an open arrowhead.

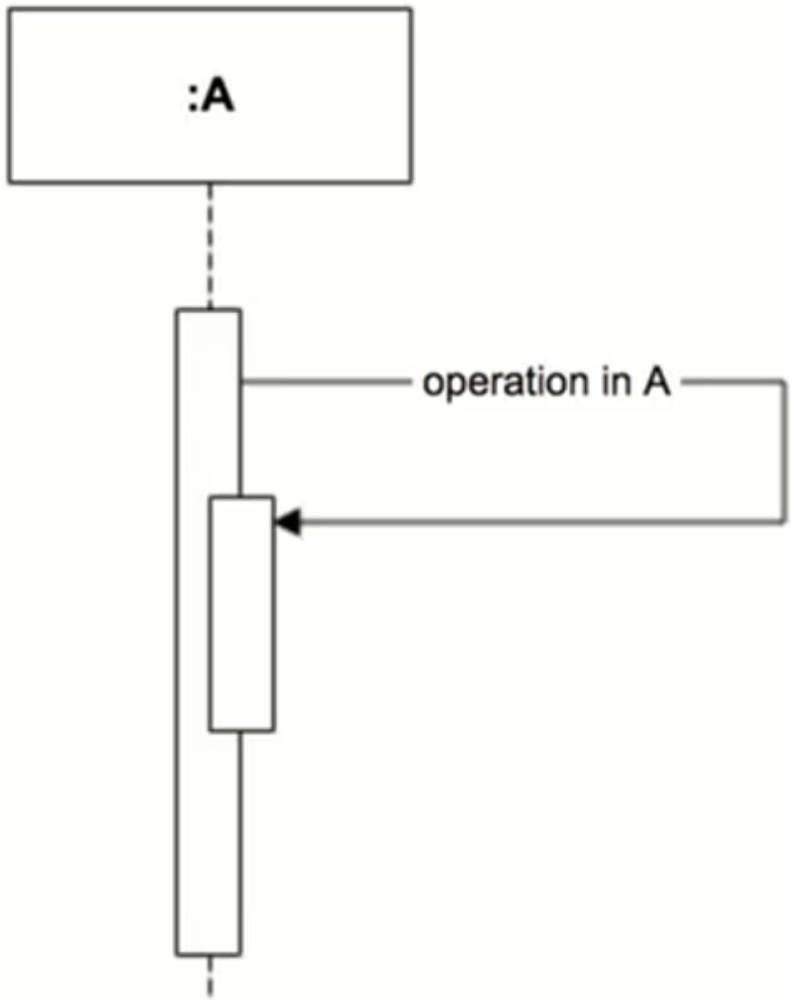
Asynchronous



Reply

- When the operation is called (function/method), the caller waits for an answer.
- In sequence diagrams this is shown with a dashed line and an open arrowhead from the “callee” to the caller.
- These are sometimes optional, but for clarity it’s good to show that control is being passed back to the caller.





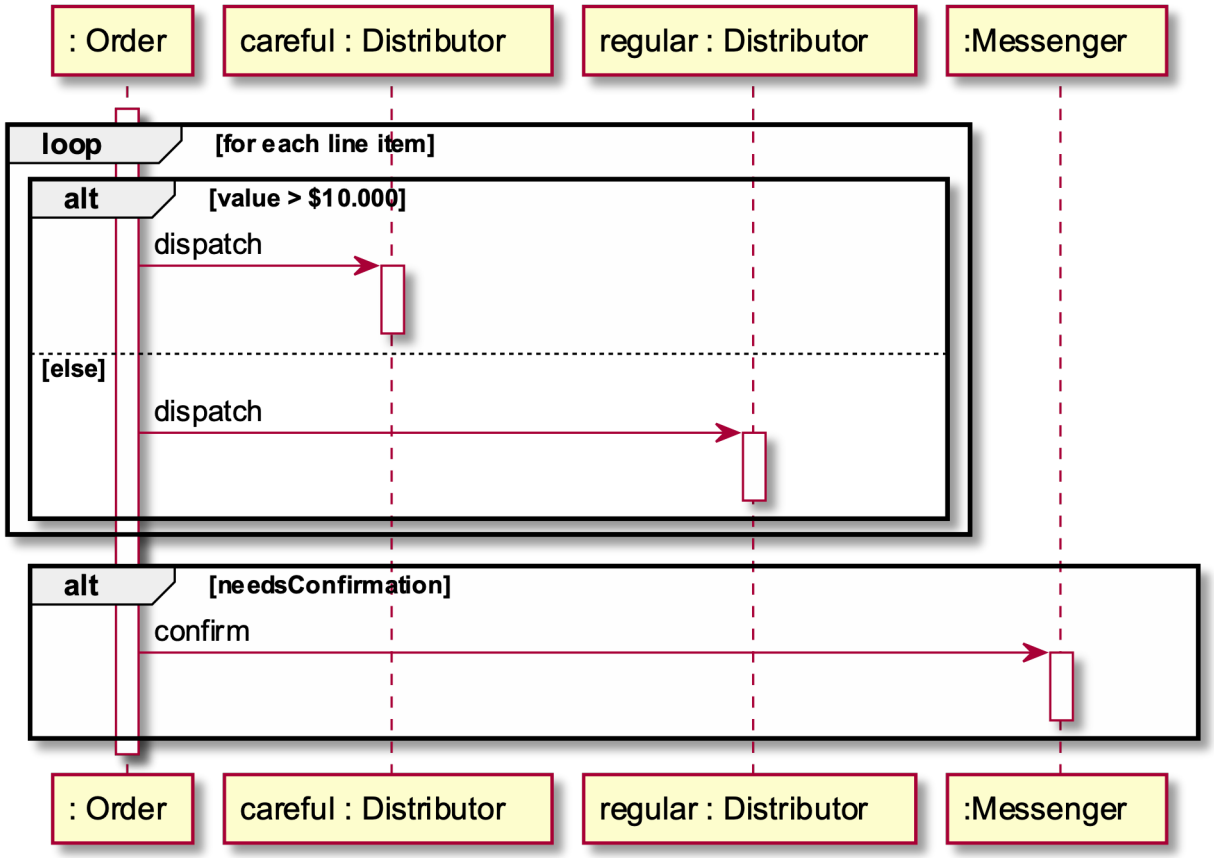
Self calls

- Sometimes, an object will call themselves. Such as when calling a method define in the same class.
- This is shown in a new “active” box placed on top of the current one.

Interaction frames

Interaction frames

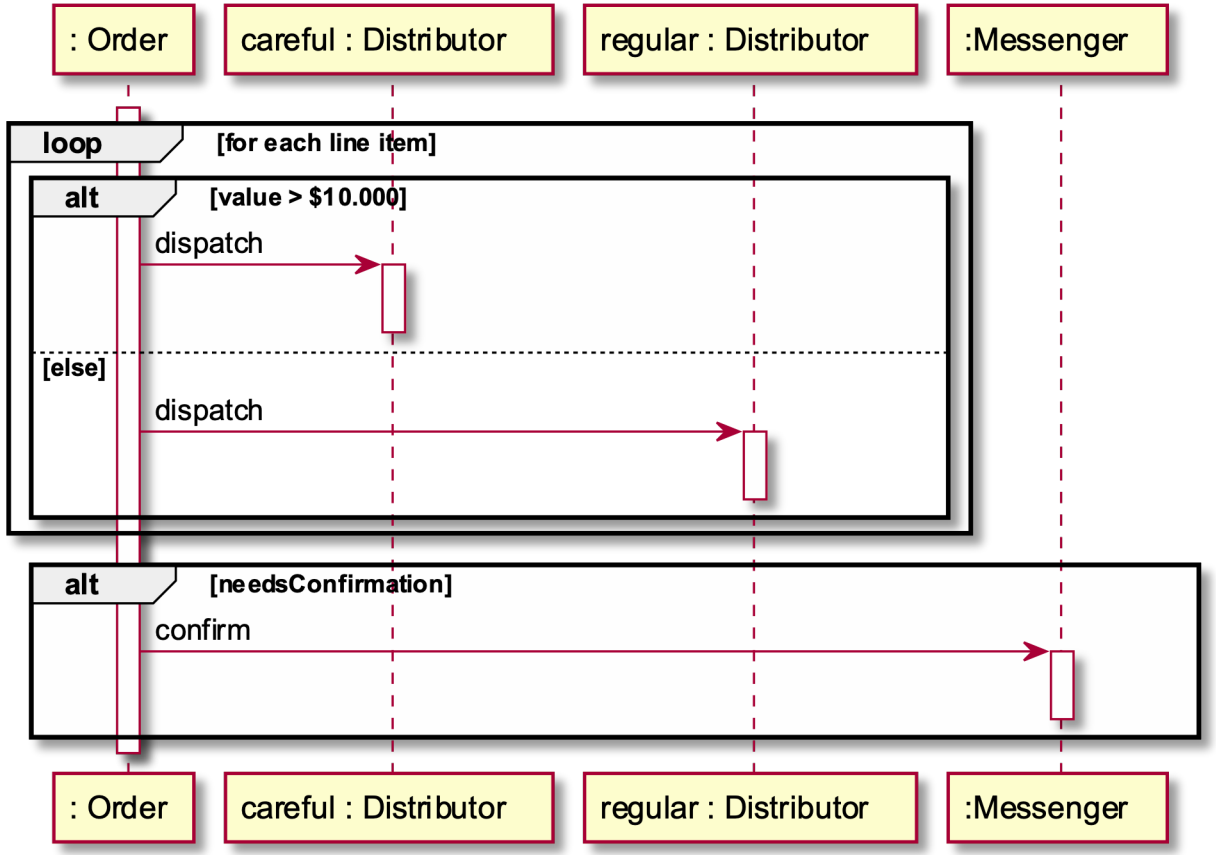
- Sequence diagrams provide us with so called interaction frames.
- These interaction frames can be used to shown *alternative sequences* in the diagram, *optional sequences*, *parallel sequences*, *repeated sequences (loops)*, and *referred sequences*.



Plant UML code

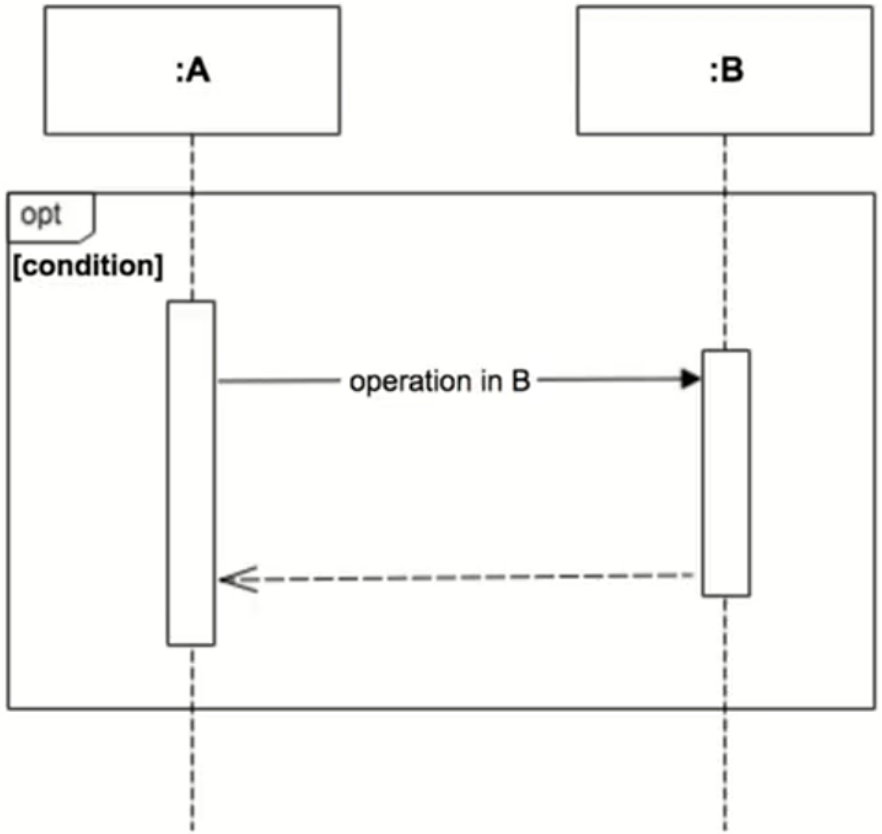
```
@startuml
participant ": Order" as order
participant "careful : Distributor" as careful
participant "regular : Distributor" as regular
participant ":Messenger" as messenger

activate order
loop for each line item
  alt value > $10.000
    order -> careful : dispatch
    activate careful
    deactivate careful
  else else
    order -> regular : dispatch
    activate regular
    deactivate regular
  end
end
alt needsConfirmation
  order -> messenger : confirm
  activate messenger
  deactivate messenger
end
deactivate order
@enduml
```



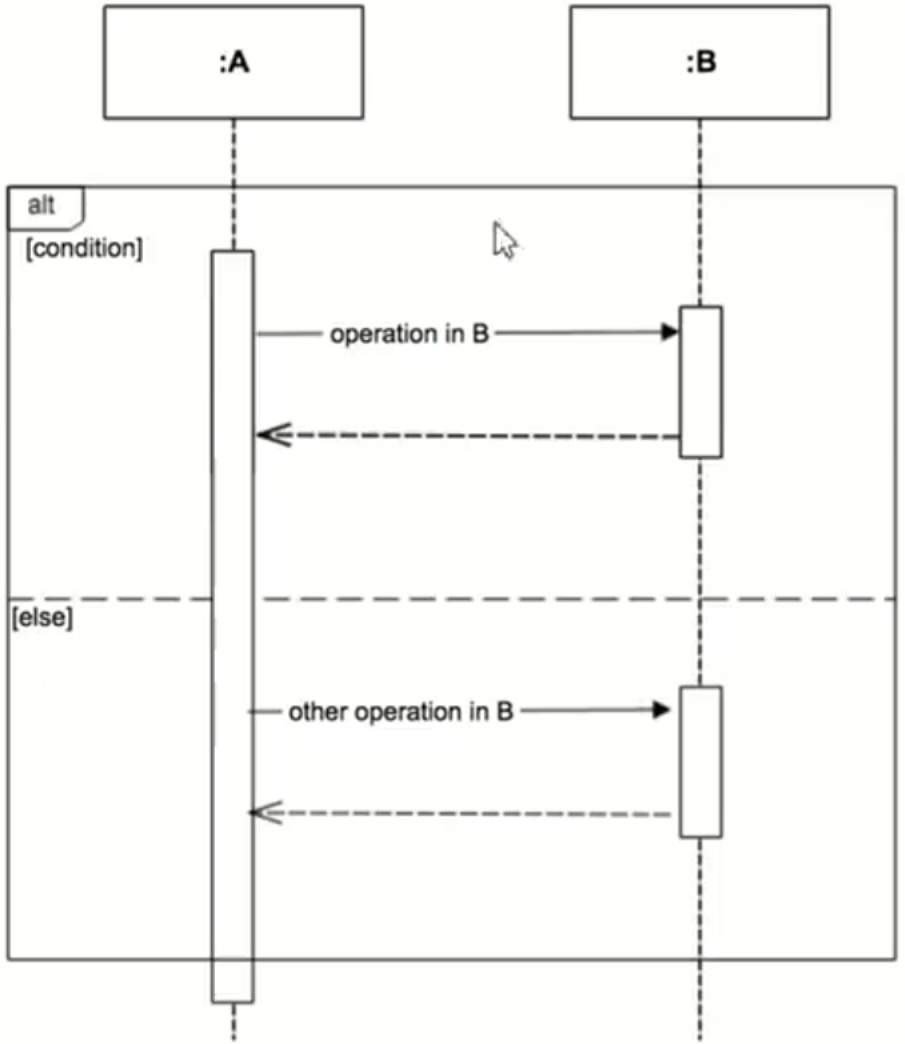
Interaction frames, contd.

- These interaction frames are really powerful, but might complicate the diagram.
- Sometimes there are other ways to demonstrate the complexity.



Optional fragments

- Optional fragments, are marked with the surrounding box, and the abbreviation “opt” in the top left corner.
- The condition needs to be met in order for the interaction frame to be activated.
- Condition can be a Boolean expression, or a sentence describing what would be the reason the optional interaction should be taken.

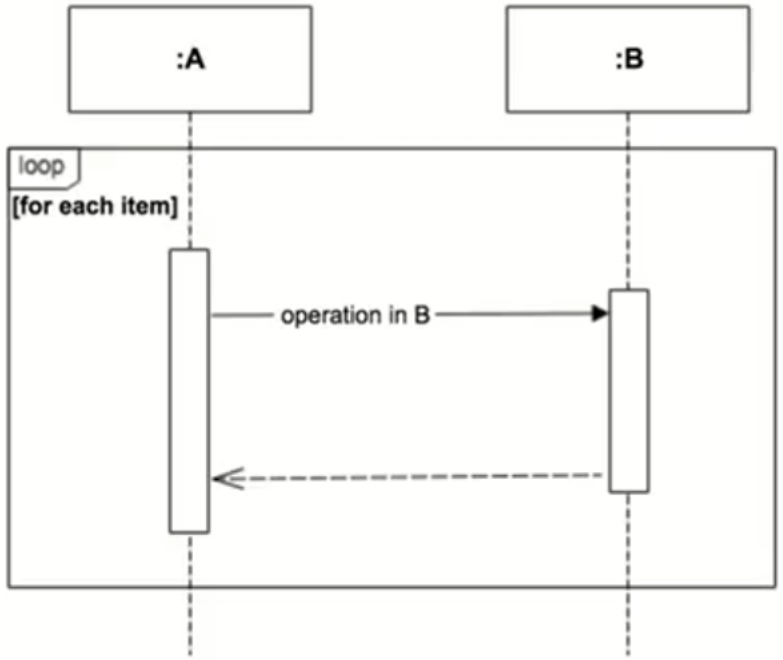


Alternative multiple fragments

- Alternative multiple fragments show two or more different frames where only one will be used.
- These must handle each case needed, that is there should be only one and always one part of the alternatives that will be run.

Repeating fragments (loops)

- The fragment may execute multiple times, but always at least once.
- Can be mixed with an optional interaction frame, and others.
- The title for the frame should show the logic behind the loop
  - for(int i=0; i <= 9; i++)
  - for student in course
  - while assignments not graded



Common operators for interaction frames

Operator	Meaning
`alt`	Alternative multiple fragments; only the one whose condition is true will execute (Figure 4.4).
`opt`	Optional; the fragment executes only if the supplied condition is true. Equivalent to an `alt` with only one trace (Figure 4.4).
`par`	Parallel; each fragment is run in parallel.
`loop`	Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration (Figure 4.4).
`region`	Critical region; the fragment can have only one thread executing it at once.
`neg`	Negative; the fragment shows an invalid interaction.
`ref`	Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
`sd`	Sequence diagram; used to surround an entire sequence diagram, if you wish.


HÁSKÓLINN Í REYKJAVÍK
REYKJAVÍK UNIVERSITY