

# Sequence Diagrams



Dr. Jason Barron

South East Technological University

January 9, 2023

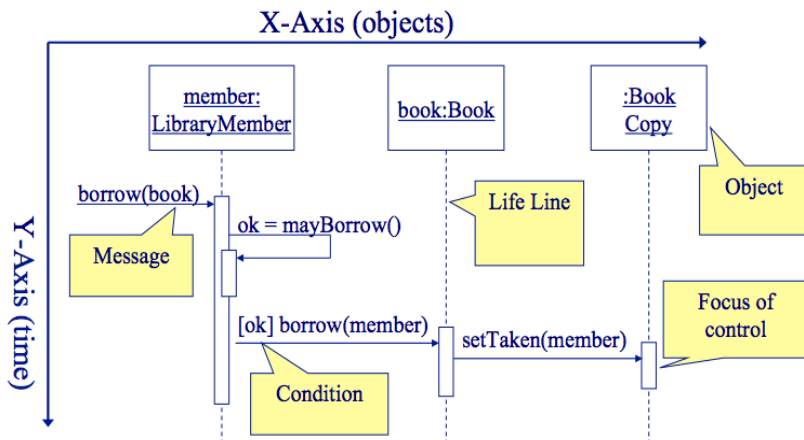
- Interaction diagrams model the behaviour of use cases by describing the way groups of objects interact to complete the task of the use case
- They portray the interaction among the objects of a system and describe the dynamic behaviour of the system
- There are two types of interaction diagrams
  - Sequence Diagrams
  - Communication Diagrams (formally known as collaboration diagrams)

- Sequence diagrams
  - Generally show the sequence of events that occur
- Collaboration diagrams
  - Demonstrate how objects are statically connected
- Both diagrams are relatively simple to draw and contain similar elements







- Purpose of interaction diagrams
  - Model interactions between objects
  - Assist in understanding how a system (i.e., a use case) actually works
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations and assign them to classes

- Illustrates the objects that participate in a use case and the messages that pass between them over time for one use case
- In design, used to distribute use case behaviour to classes

# Sequence Diagram



# Sequence Diagram Syntax

AN ACTOR	
AN OBJECT	
A LIFELINE	
A FOCUS OF CONTROL	
A MESSAGE	
OBJECT DESTRUCTION	

- Two major components:
  1. Active objects
  2. Communication between these active objects (i.e messages sent between the active objects)

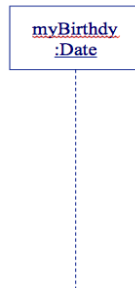


- Active objects
  - Any objects that play a role in the system
  - Participate by sending and/or receiving messages
  - Placed across the top of the diagram
  - Can be:
    - An actor (from the use case diagram)
    - Object/class (from the class diagram) within the system

- Object
  - Can be any object or class that is valid within the system
  - Object naming
    - Syntax

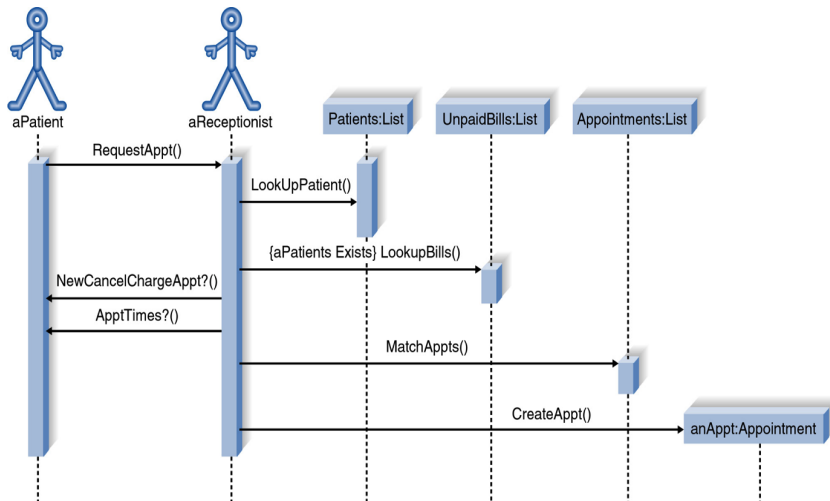
[instanceName][:className]

1. Class name only: **:Classname**
2. Instance name only:  
**objectName**
3. Instance name and class name together: **object:Class**







- Actor
  - A person or system that derives benefit from and is external to the system
  - Participates in a sequence by sending and/or receiving messages

# Sequence Diagram Example

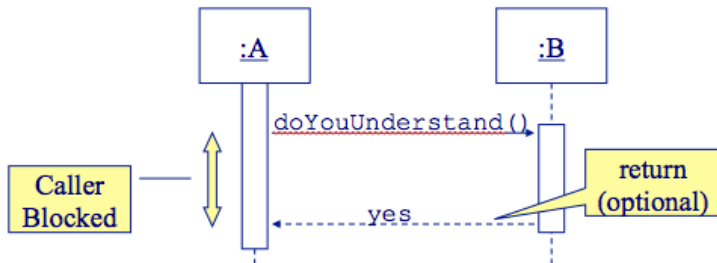


- Messages
  - Used to illustrate communication between different active objects of a sequence diagram
  - Used when an object needs
    - to activate a process of a different object
    - to give information to another object

- A message is represented by an arrow between the life lines of two objects
  - Self calls are allowed
- A message is labelled at minimum with the message name
  - Arguments and control information (conditions, iteration) may be included

Synchronous (flow interrupt until the message has completed)	
Asynchronous (dont wait for response)	
Flat (no distinction between syn/asyn)	
Return (control flow has returned to the caller)	

- The routine that handles the message is completed before the calling routine resumes execution





- Calling routine does not wait for the message to be handled before it continues to execute
  - As if the call returns immediately
- Examples
  - Notification of somebody or something
  - Messages that post progress information

- Optionally indicated using a dashed arrow with a label indicating the return value
  - Don't model a return value when it is obvious what is being returned, e.g. **getTotal()**
  - Model a return value only when you need to refer to it elsewhere (e.g. as a parameter passed in another message)
  - Prefer modelling return values as part of a method invocation, e.g. **ok = isValid()**



- Lifeline
- Focus of control (activation box or execution occurrence)
- Control information
  - Condition, repetition

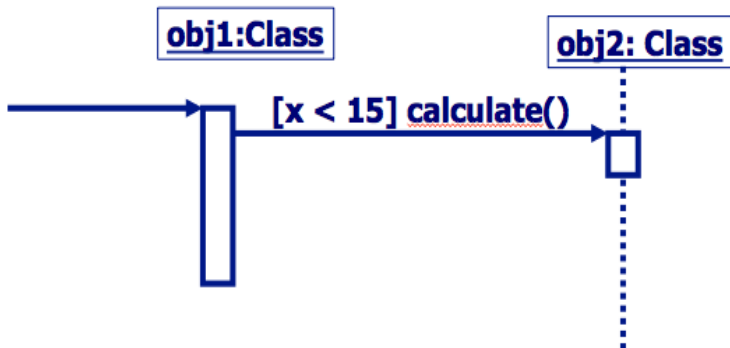
- Lifeline
  - Denotes the life of actors/objects over time during a sequence
  - Represented by a vertical line below each actor and object (normally dashed line)
- For temporary object
  - Place X at the end of the lifeline at the point where the object is destroyed

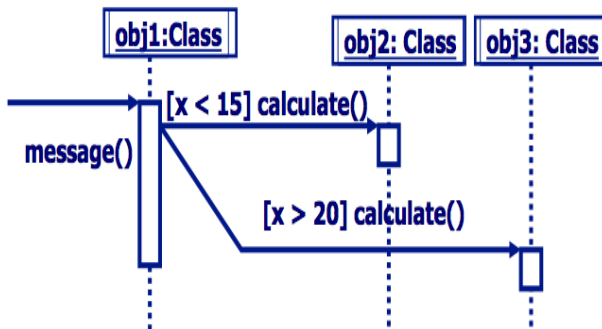
- Focus of control (activation box)
  - Means the object is active and using resources during that time period
  - Denotes when an object is sending or receiving messages
  - Represented by a thin, long rectangular box overlaid onto a lifeline

- Condition
  - syntax: *'[expression]'message — label*
  - The message is sent only if the condition is true

[ok] borrow(member)

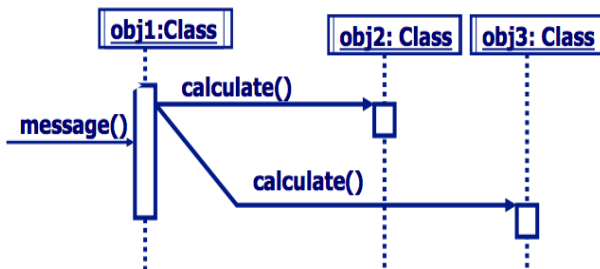
# Elements of Sequence Diagram







## Concurrency



- Control information
  - Iteration
    - May have square brackets containing a continuation condition (until) specifying the condition that must be satisfied in order to exit the iteration and continue with the sequence
    - May have an asterisk followed by square brackets containing an iteration (while or for) expression specifying the number of iterations

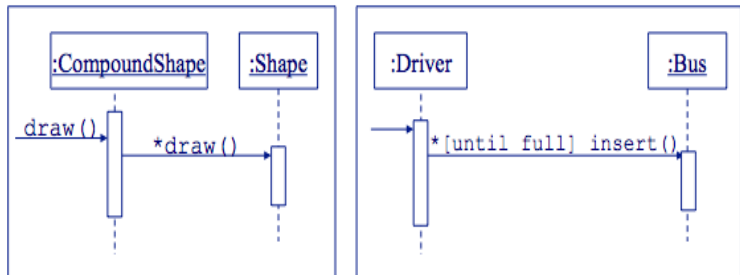
- Iteration
  - syntax: `*['[expression']']message – label`
  - The message is sent many times to possibly multiple receiver objects

**\*draw()**



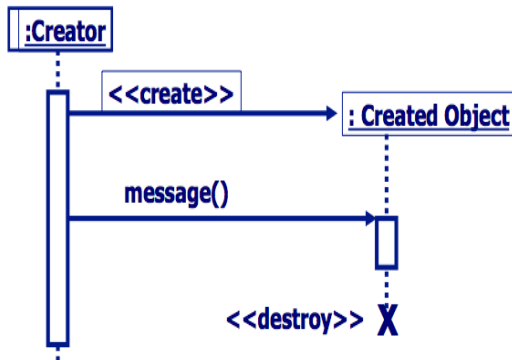
# Control Information

## Iteration Example



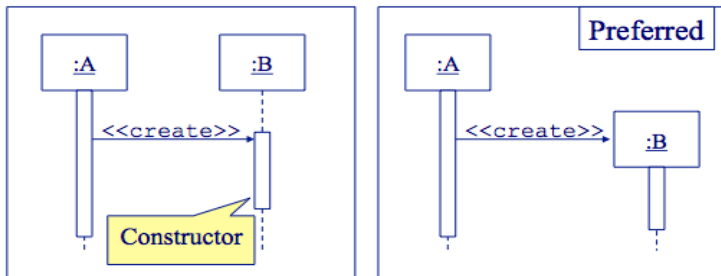
- The control mechanisms of sequence diagrams suffice only for modelling simple alternatives
  - Consider drawing several diagrams for modelling complex scenarios
  - Don't use sequence diagrams for detailed modelling of algorithms (this is better done using activity diagrams, pseudo-code or state-charts)

- Creation and destruction of an object in sequence diagrams are denoted by the stereotypes `<< create >>` and `<< destroy >>`



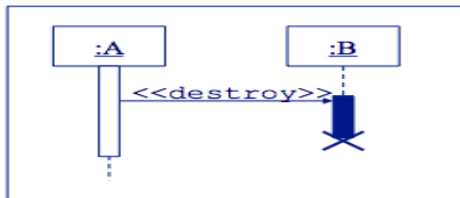
- Notation for creating an object on-the-fly
  - Send the `<< create >>` message to the body of the object instance
  - Once the object is created, it is given a lifeline
    - Now you can send and receive messages with this object as you can any other object in the sequence diagram

- An object may create another object via a `<< create >>` message

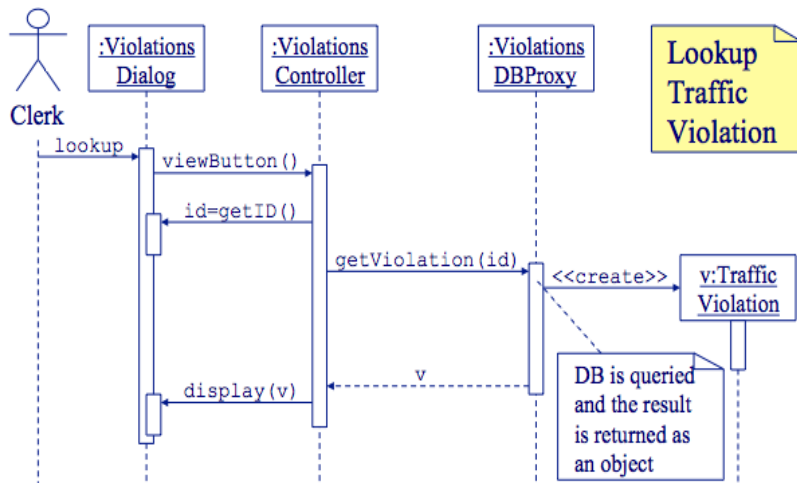




- An object may destroy another object via a `<<destroy>>` message
  - An object may destroy itself
  - Avoid modelling object destruction unless memory management is critical



# Sequence Diagram Example



1. Set the context
2. Identify which objects and actors will participate
3. Set the lifeline for each object/actor
4. Lay out the messages from the top to the bottom of the diagram based on the order in which they are sent
5. Add the focus of control for each objects or actor's lifeline
6. Validate the sequence diagram

# 1. Set the Context

1. Select a use case
2. Decide the initiating actor

## 2. Identify the Objects

- Identify the objects that may participate in the implementation of this use case by completing the supplied message table
- 1. List candidate objects
  - 1.1. Use case controller class
  - 1.2. Domain classes
  - 1.3. Database table classes
  - 1.4. Display screens or reports
- 2. List candidate messages (in message analysis table)
  - 2.1. Examine each step in the normal scenario of the use case description to determine the messages needed to implement that step
  - 2.2. For each step:
    - 2.2.1. Identify step number
    - 2.2.2. Determine messages needed to complete this step
    - 2.2.3. For each message, decide which class holds the data for this action or performs this action
  - 2.3. Make sure that the messages within the table are in the same order as the normal scenario
- 3. Begin sequence diagram construction
  - 3.1. Draw and label each of the identified actors and objects across the top of the sequence diagram
  - 3.2. The typical order from left to right across the top is the actor, primary display screen class, primary use case controller class, domain classes (in order of access), and other display screen classes (in order of access)

### 3. Set the Lifeline

- Set the lifeline for each object/actor

## 4. Lay out Messages

- Lay out the messages from the top to the bottom of the diagram based on the order in which they are sent
  1. Working in sequential order of the message table, make a message arrow with the message name pointing to the owner class
  2. Decide which object or actor initiates the message and complete the arrow to its lifeline
  3. Add needed return messages
  4. Add needed parameters and control information

## 5. Add Focus of Control

- Add the focus of control (activation box) for each objects or actors lifeline



## 6. Validate Diagram

- Validate the sequence diagram

- [1] Ivar Jacobsen, Grady Booch & James Rumbaugh, 'The Unified Software Development Process', Addison Wesley, 1999.