SEQUENCE DIAGRAM

Interaction Diagrams

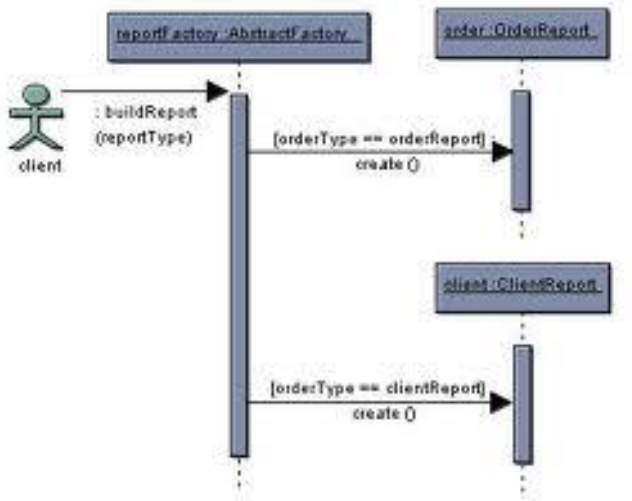
- UML Specifies a number of interaction diagrams to model dynamic aspects of the system
- Dynamic aspects of the system
    - Messages moving among objects/classes
    - Flow of control among objects
    - Sequences of events

Sequence Diagrams

- Illustrates how objects interacts with each other.
- Emphasizes time ordering of messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.

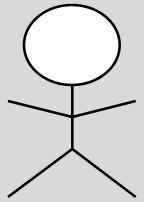
Sequence Diagrams

- Describe the flow of messages, events, actions between objects



- Show concurrent processes and activations
- Show time sequences that are not easily depicted in other diagrams
- Typically used during analysis and design to document and understand the logical flow of your system

Sequence Diagram Syntax

AN ACTOR	
AN OBJECT	anObject:aClass
A LIFELINE	
A FOCUS OF CONTROL	
A MESSAGE	aMessage()
OBJECT DESTRUCTION	X

Sequence Diagram

- **Two major components**
  - **Active objects**
  - **Communications between these active objects**
    - *Messages sent between the active objects*

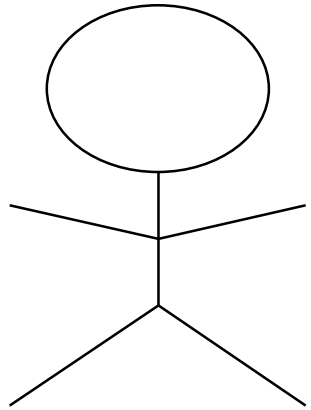
Sequence Diagram

- 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

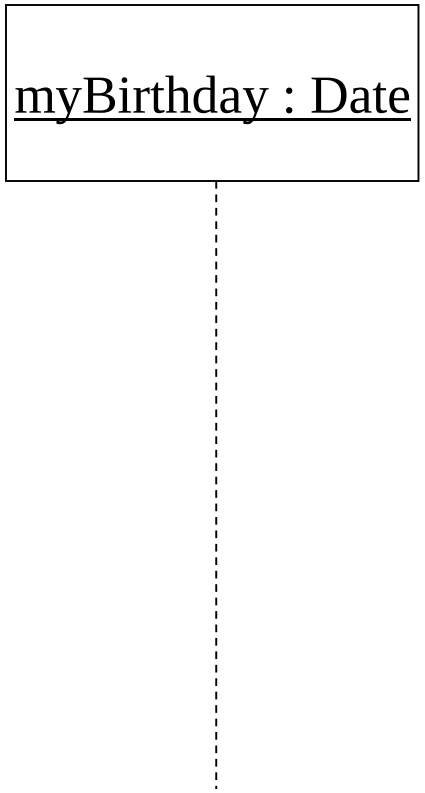
Active Objects

- 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



Object

- Object naming:
  - syntax: [instanceName][:className]
  - Name classes consistently with your class diagram (same classes).
  - Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.
- The Life-Line represents the object's life during the interaction



Messages

- An interaction between two objects is performed as a message sent from one object to another (simple operation call, Signaling, RPC)
- If object obj_1 sends a message to another object obj_2 some link must exist between those two objects (dependency, same objects)

Types of Messages

- Synchronous (flow interrupt until the message has completed)

- Asynchronous (don’t wait for response)

- Flat (no distinction between sync/async)

- Return (control flow has returned to the caller)

Messages (Cont.)

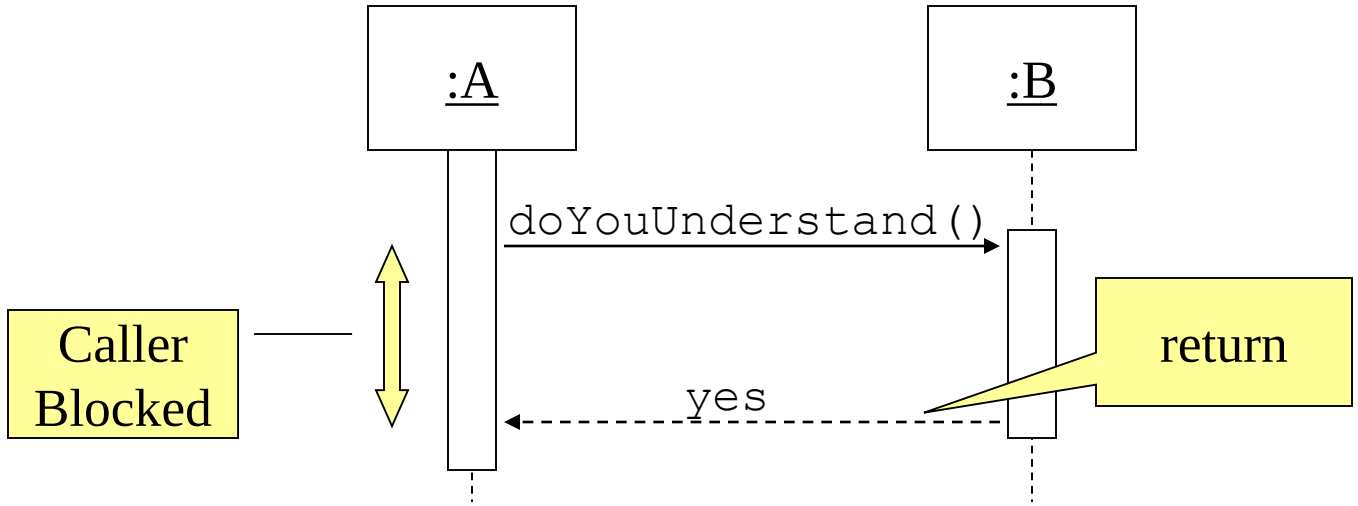
- A message is represented by an arrow between the life lines of two objects.
  - Self calls are also allowed
  - The time required by the receiver object to process the message is denoted by an activation-box.
- A message is labeled at minimum with the message name.
  - Arguments and control information (conditions, iteration) may be included.

Return Values

- 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 modeling return values as part of a method invocation, e.g. **`ok = isValid()`**

Synchronous Messages

- Nested flow of control, typically implemented as an operation call.
  - The routine that handles the message is completed before the caller resumes execution.



Asynchronous Message

- Used when we don’t want the sender to wait for a response
  - Typically a one way message
  - No response is sent.
  - Response may invoke a callback method.

Flat Messages

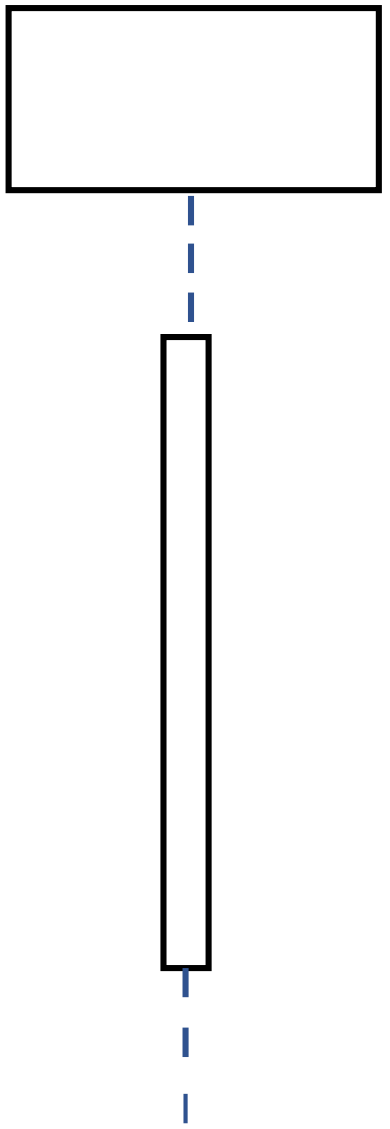
- Used when we don’t want to specify whether or not sender waits for a response.
  - Haven’t decided yet.
  - Isn’t important
  - Specifically want to leave as an implementation decision.

Sequence Diagram

- 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 object
  - place X at the end of the lifeline at the point where the object is destroyed

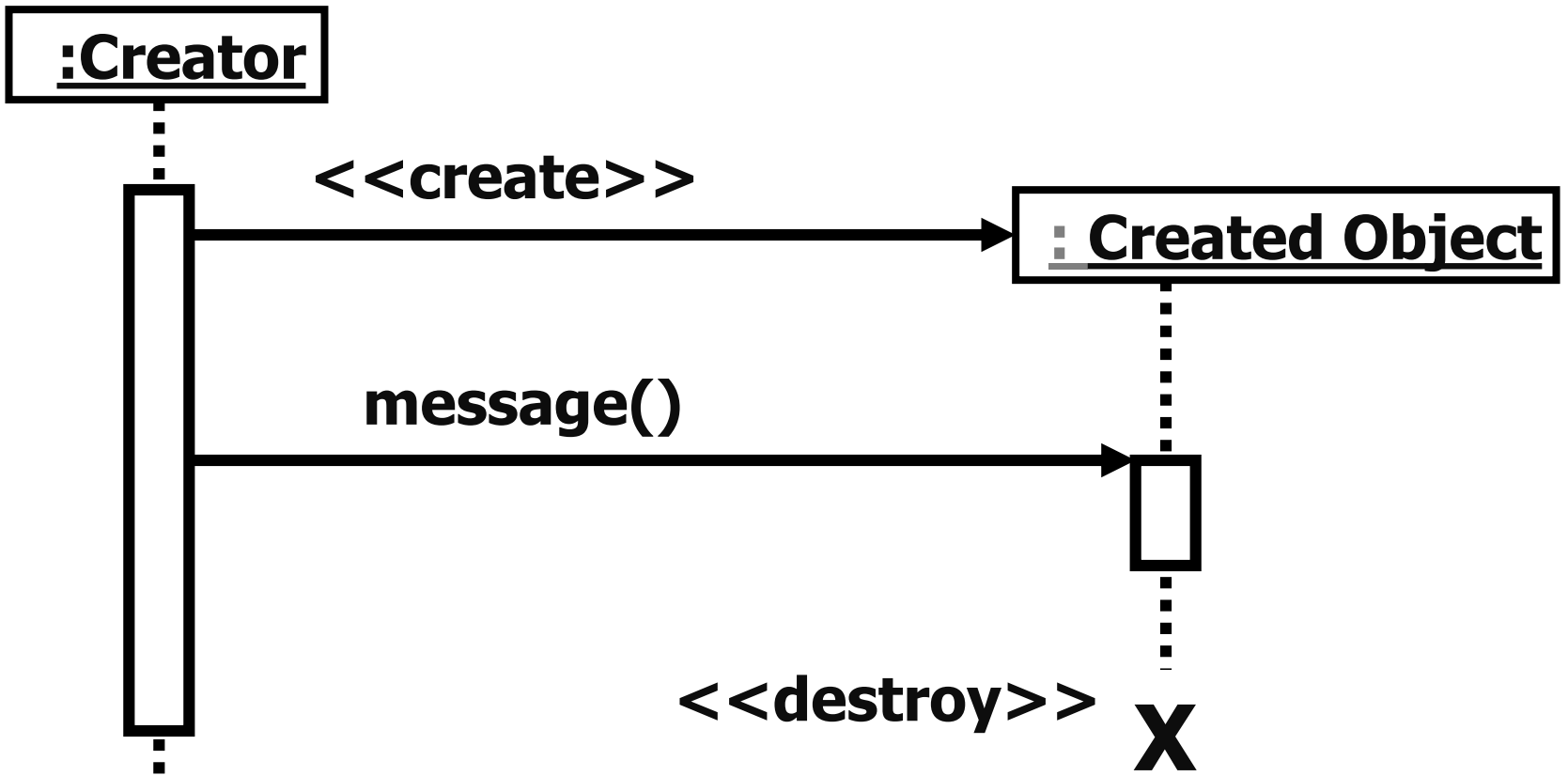
Sequence Diagram

- 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



Sequence Diagrams

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

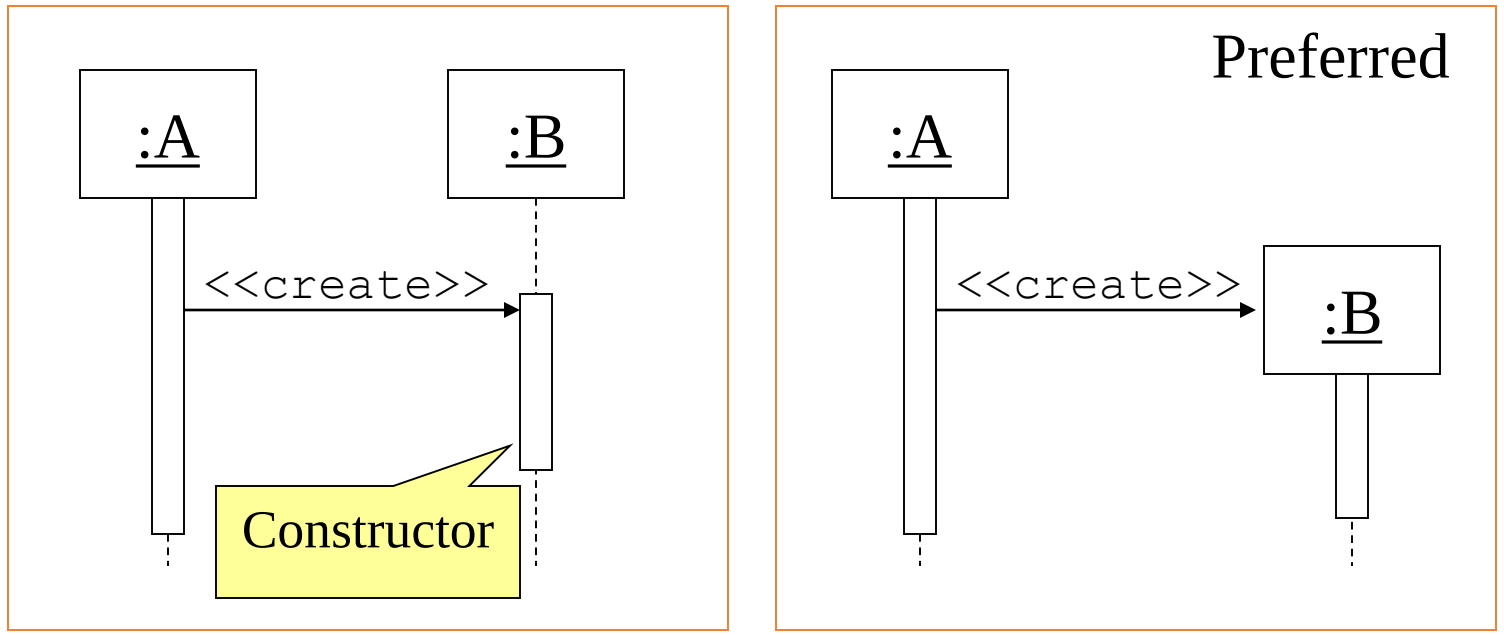


Creating Objects

- 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.

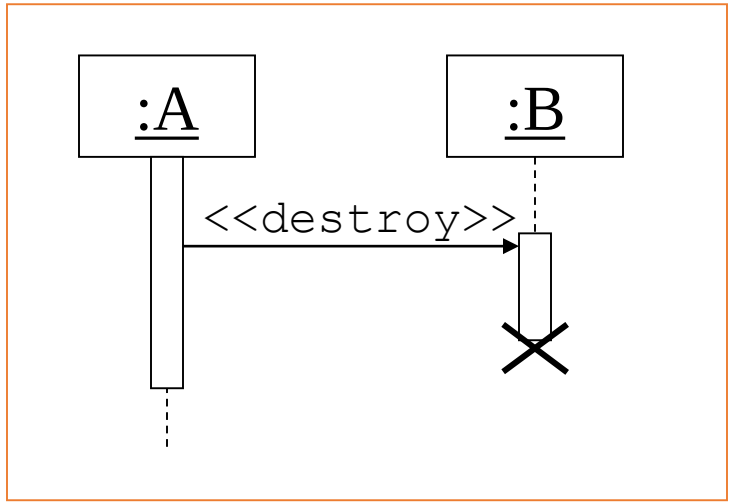
Object Creation

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

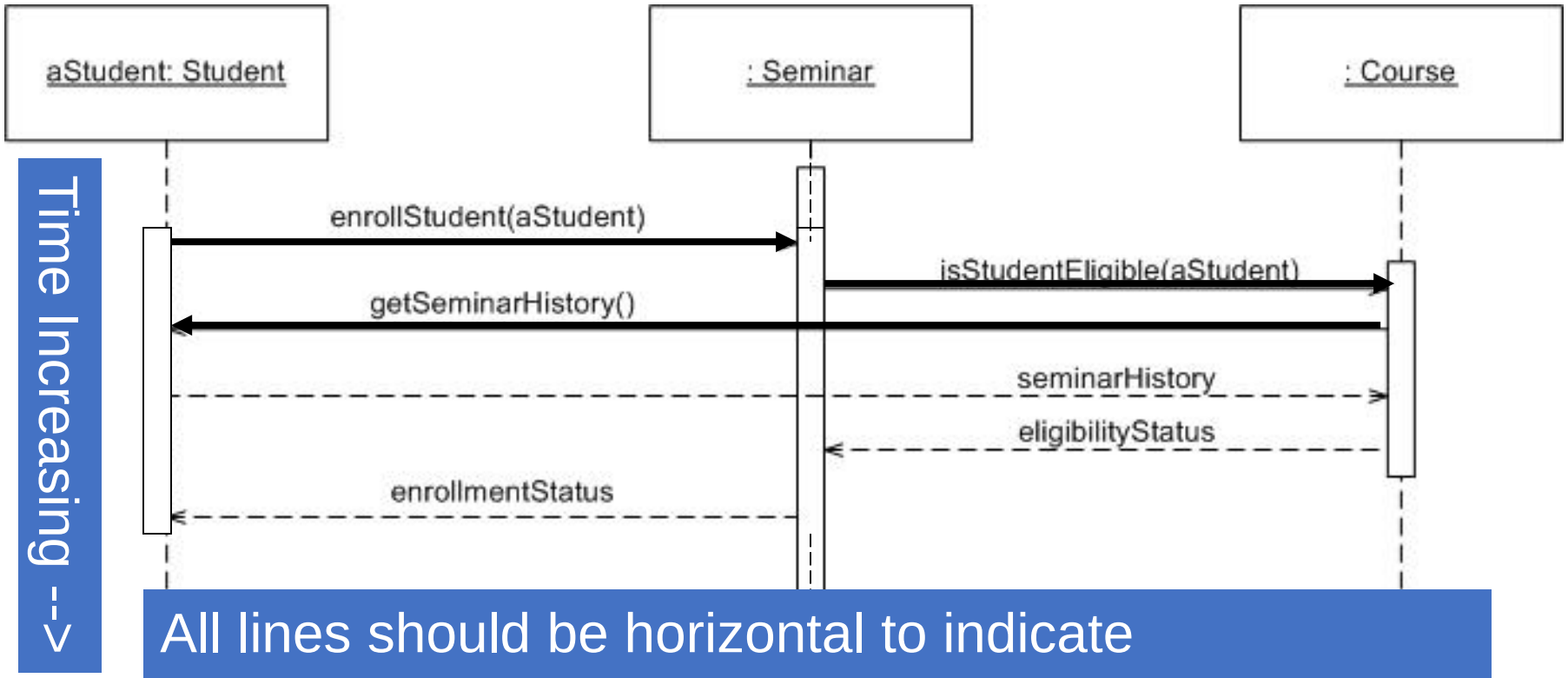


Object Destruction

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



Sequence Diagram



All lines should be horizontal to indicate instantaneous actions. Additionally if ActivityA happens before ActivityB, ActivityA must be above activity A

Steps for Building a Sequence Diagram

- Set the context
- Identify which objects and actors will participate
- Set the lifeline for each object/actor
- Lay out the messages from the top to the bottom of the diagram based on the order in which they are sent
- Add the focus of control for each object’s or actor’s lifeline
- Validate the sequence diagram