CSE 308

UML
Sequence Diagrams

Reading / Reference

- Reading

www.ibm.com/developerworks/rational/library/3101.html

- Reference

www.visual-paradigm.com/VPGallery/diagrams/Sequence.html

2

Interaction Diagrams

- Sequence diagrams and collaboration diagrams
- A series of diagrams describing the *dynamic behavior* of an object-oriented system
- Often used to model a use case
- The purpose of Interaction diagrams is to:
  - Model interactions between objects
  - Verify that a use case description can be supported by the existing classes
  - Identify new classes
  - Assign responsibilities/operations to classes

We focus on sequence diagrams

3

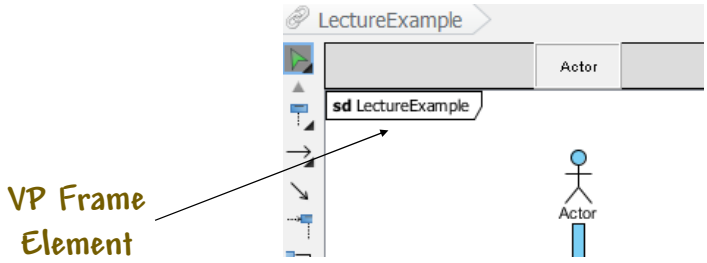
UML Sequence Diagram

- Sequence diagram - an interaction diagram that models a single scenario (use case) executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)
- Illustrates how objects interact with each other
- Emphasizes time ordering of messages
- Can model simple sequential flow, branching, iteration, recursion and concurrency

4

Frame Element

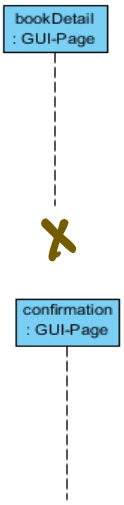
- The graphical boundary of a diagram
- Provides a consistent place for a diagram's label
- Provides a graphical boundary for the diagram
- Optional in UML diagrams



5

Lifeline

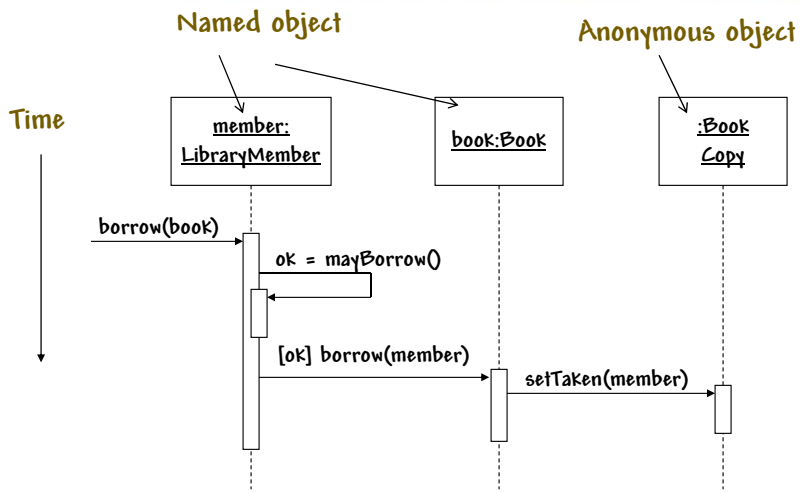
- Think of a lifeline as a "live" object
- Lifelines represent either roles or object instances
- An "x" is shown when the object is destroyed
- Placement
  - usually across the top of the diagram
  - Depending on tool, you might lower the placement of the lifeline if object activation occurs during the use case



6

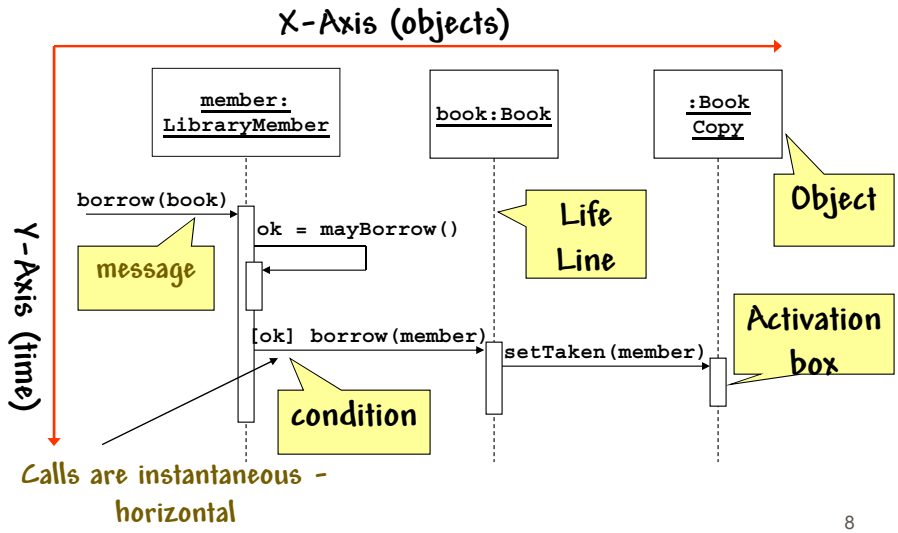
Sequence Diagram Syntax

Not a realistic example



7

A Sequence Diagram



8

Key Components

- Participant: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached arrow or an arrow attached to an actor

In a GUI system the initial participant is usually an actor

- Message: communication between objects/actors
- Axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)

9

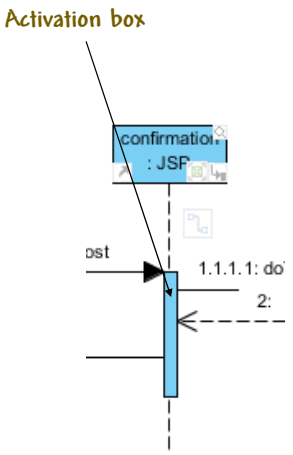
Messages

- An interaction between two objects is performed as a message sent from one object to another (e.g., method call)
- If an object sends a message to another object, object 1 must have visibility to object 2 (i.e., have a handle)
- 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 a minimum with the message name

10

Indicating Method Calls

- Activation box: thick box over object's life line; drawn when object is on the stack
  - Either that object is running its code, or it is on the stack waiting for another object's method to finish
  - Nest to indicate recursion



11

Messages

- Solid arrow heads represent synchronous calls
  - a synchronous message waits until the message is done (e.g., invoking a subroutine)
- Open arrow heads represent asynchronous messages
  - An asynchronous message can continue processing and doesn't have to wait for a response
  - Example: Ajax calls from GUI
- Dashed lines represent reply messages.

12

Arrow Labels

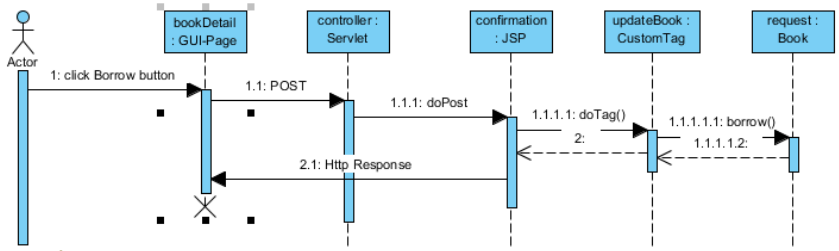
- Method call
  - Label the call arrow with the method name
  - Include parameters if they are not obvious
- Return
  - Don't model a return value when it is obvious what is being returned, e.g. getTotal()
  - Model a return value only when you need to refer to it elsewhere, e.g. as a parameter passed in another message

In general, don't model obvious interactions if you are not intending to use the generated code

13

VP Example

- Example below is incomplete, but it shows the use of VP for one approach to a typical project use case



Some Missing items

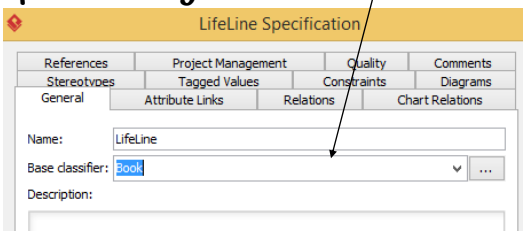
- Update of persistence layer
- Visibility of objects (e.g., no Session object)
- Connection to download
- Activation of new GUI page
- Update of other objects/properties (e.g., licenses)

Alternate approaches might update the server with an Ajax call or in a distinct controller for Borrow

14

Visual Paradigm Hints

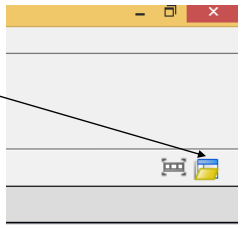
- Include your class diagrams and sequence diagrams in a project
- Always (almost?) select your class in a sequence diagram from the known classes in your class diagram. VP allows you to easily toggle between class and sequence diagrams.



15

Realistic Design Approach

- Use your sequence diagrams to identify classes and class attributes needed in your class diagram
- Work both simultaneously (e.g. add methods to your class diagram once you see that you need it)
- Don't be reluctant to modify your design during this stage
- Use VP button to easily go back and forth between class diagram and sequence diagram



16

Project Team Approach

- The first few diagrams will be very difficult to do
- Do the first few as a team (with lots of team interaction)
- Once your team begins to understand your design philosophy and framework philosophy, you will be able to assign parts to team members
- Look for common design approaches (e.g., DB access, server access, session management), you might be able to use sub-diagrams

17

Project Hints

- Be sure to show an understanding of the object in your GUI (i.e., browser based)
- Concentrate on backend logic
  - GUI object interaction will vary based on your choice of development framework
  - Generalize the DB component in your initial sequence diagrams (e.g., just show a general DB call from a persistence layer object)
  - Develop one generalized sequence diagram for GUI call and DB call once you understand your approach to each

18

GUI Approaches

- Native servlets/JSPs
  - Independent controller servlets
- Framework (e.g., Spring)

Note that the object calls above are form different environments

19

Design Review

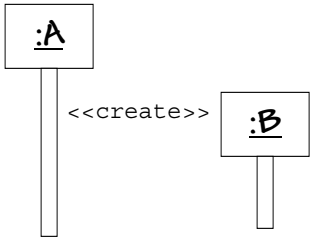
- Design review will be organized along the lines of use cases (and corresponding sequence diagrams)
- Your team gets to pick the first use case to show (try not to use login)
- Clarity of thinking and consistency are more important than getting the best possible design approach

20

Object Instantiation

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

Preferred

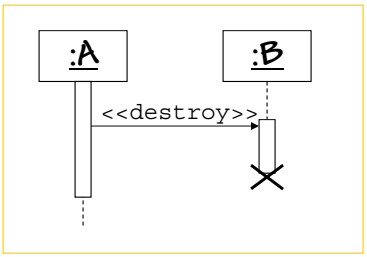


Using new is OK, but you will be probably use the factory design pattern

21

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.



22

Indicating Selection and Loops

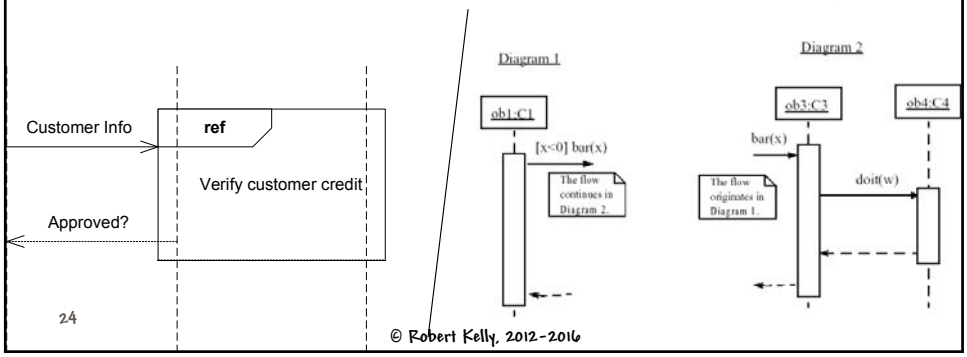
- frame: box around part of a sequence diagram to indicate selection or loop
  - if -> (opt) [condition]
  - if/else -> (alt) [condition], separated by horiz. dashed line
  - loop -> (loop) [condition or items to loop over]

23

Linking Sequence Diagrams

- If one sequence diagram is too large or refers to another diagram, indicate it with either:
  - An unfinished arrow and comment
  - A "ref" frame that names the other diagram

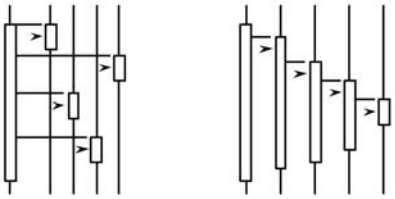
Although this might result from the use case being too large



24

(De)centralized System Control

- What can you say about the control flow of each of the following systems?
  - centralized?
  - distributed?



25

Why Not Just Code It?

- Sequence diagrams can be somewhat close to the code level. So why not just code the algorithm rather than drawing it as a sequence diagram?
  - Allows you to think through design issues
  - A good sequence diagram is well above the level of the real code
  - Tool might generate code
  - Sequence diagrams are language-agnostic (can be implemented in many different languages)
  - Easier to do as a team
  - Can see many objects/classes at the same time

26

Sequence Diagram Exercise

- A volunteer group will select a use case from its project, and draw the corresponding use case diagram, using Violet

27