Sequence Diagram

CSE 3223

Mir Tafseer Nayeem

Faculty Member, CSE AUST

tafseer.nayeem@gmail.com

Interaction Diagram

- A series of diagram describing the *dynamic behavior* of an object oriented system.
- **Purpose of Interaction Diagram:**
  - Model interactions between objects.
  - Verify that the Use Case description can be supported by the existing classes.
  - Identify responsibilities / operations and assign them to classes.
- **Common types of Interaction Diagram:**
  - **Sequence Diagram**
  - Collaboration Diagram
  - Activity Diagram

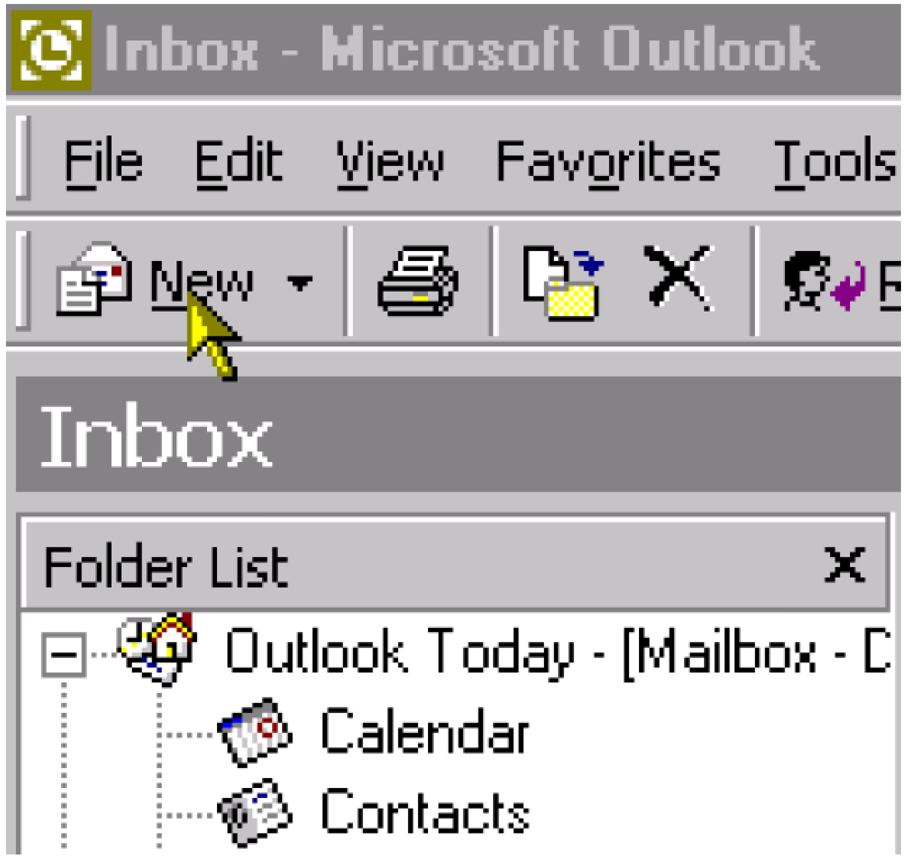
Sequence Diagram

- **Use case modeling**, which is mostly used to model interactions between a system and external actors (users or other systems).
- **Sequence diagrams**, which are used to model interactions between system components, although external agents may also be included.
- Sequence diagrams are used to model interactions between the actors and the objects in a system and the interaction between the objects themselves.
- It shows the sequence of interactions that take place during particular use case.
- Typically used to understand the logical flow of a system.

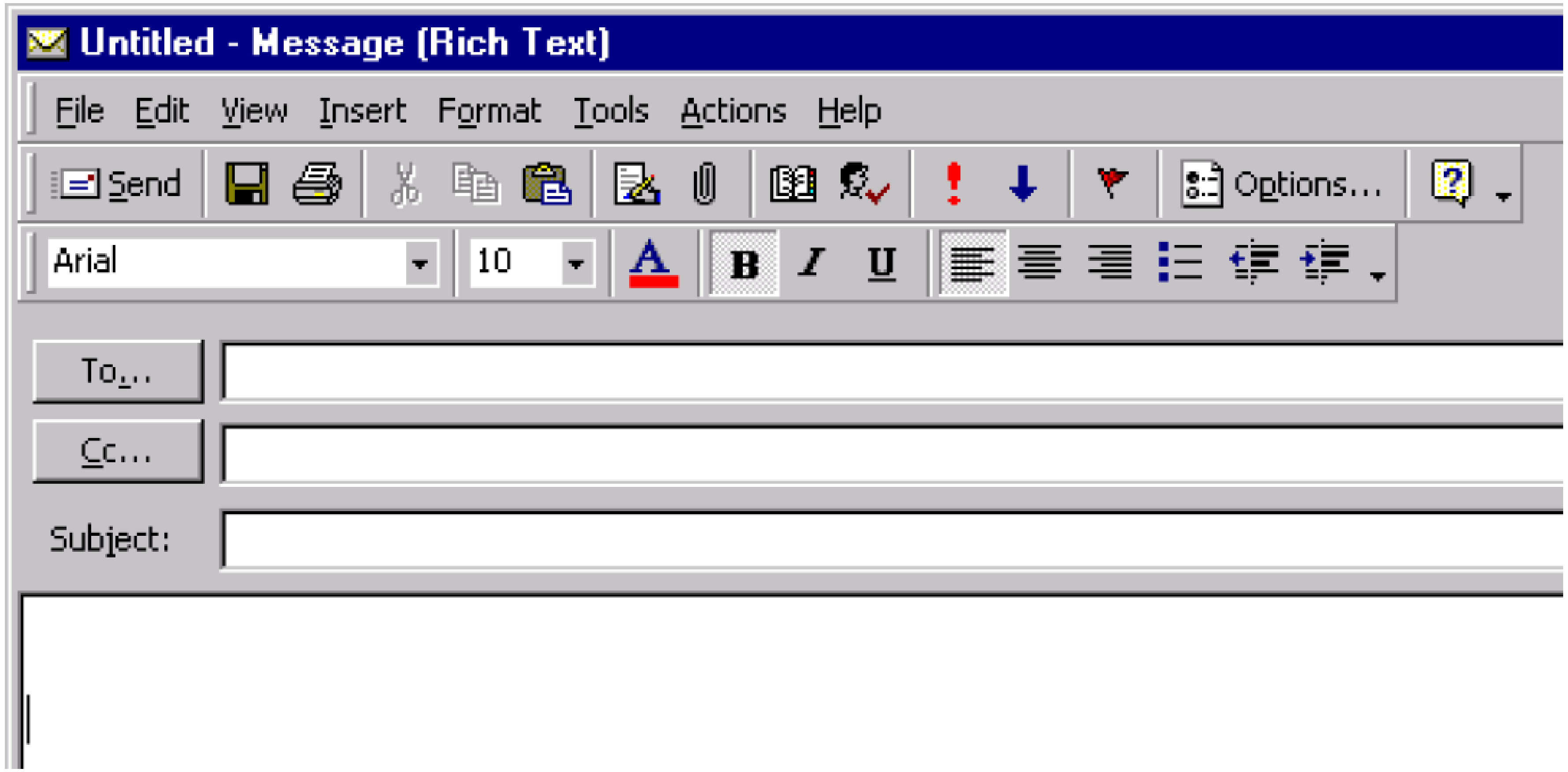
Sequence Diagram

- Shows how processes operate with one another in what order.
- Illustrates how objects interact with each other.
- Emphasizes time ordering of the messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.
- **Key parts of a Sequence Diagram:**
  - **Participant:** An object or entity that acts in the sequence diagram.
  - **Message:** Communication between the participants / objects.

A Simpler Example - Sending an email


Inbox - Microsoft Outlook
File Edit View Favorites Tools
New
Inbox
Folder List
Outlook Today - [Mailbox - C
Calendar
Contacts

E-mail Interface


Untitled - Message (Rich Text)
File Edit View Insert Format Tools Actions Help
Send
Options...
Arial
10
B
I
U
To...
Cc...
Subject:

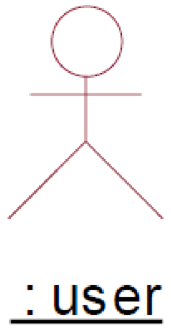
Working From a Scenario

<u>Sending an email</u>

1. Press “New ” email icon
2. Enter person’s name in “To” section
3. Type subject
4. Type contents
5. Press Send button
6. System looks up email address in address book
7. System submits the email to the email server

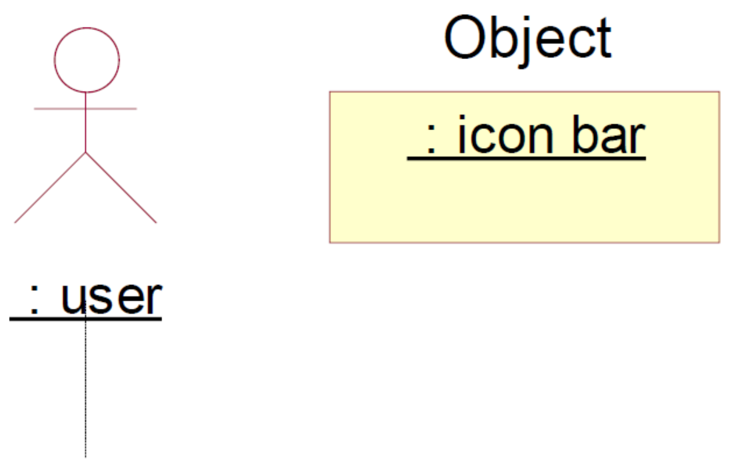
Starting The Diagram

If this is an interactive scenario, we always have an actor driving it



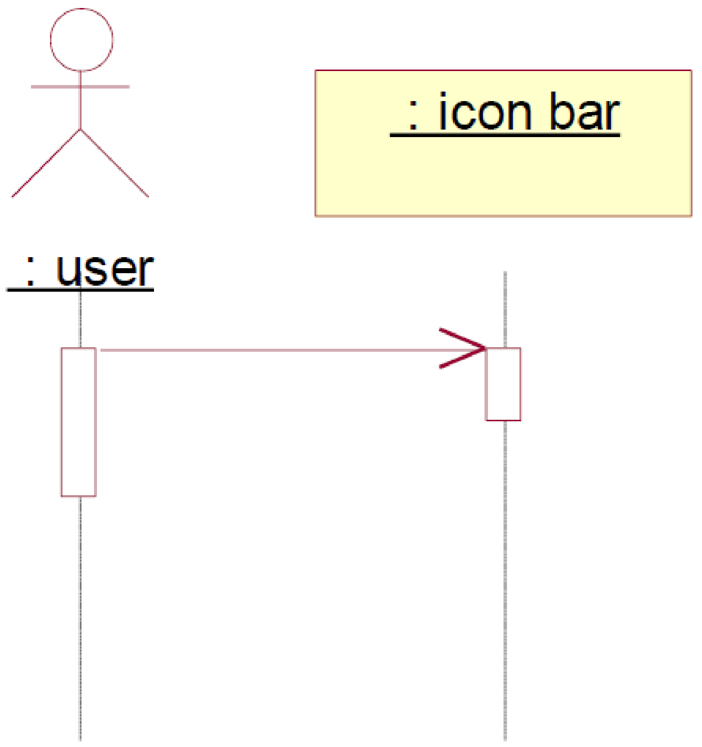
Add Objects

The first interaction is with the icon bar, which we can treat as an



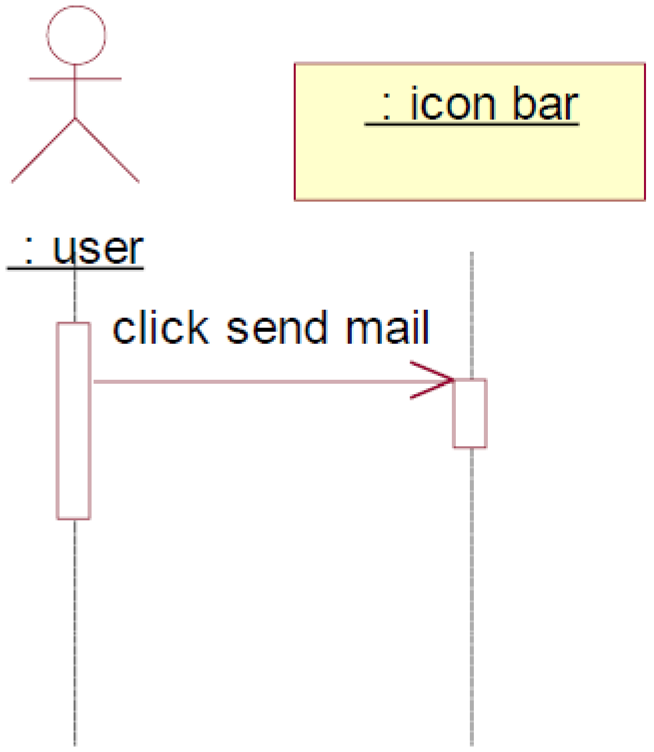
Add Message

- The user talks to the icon bar



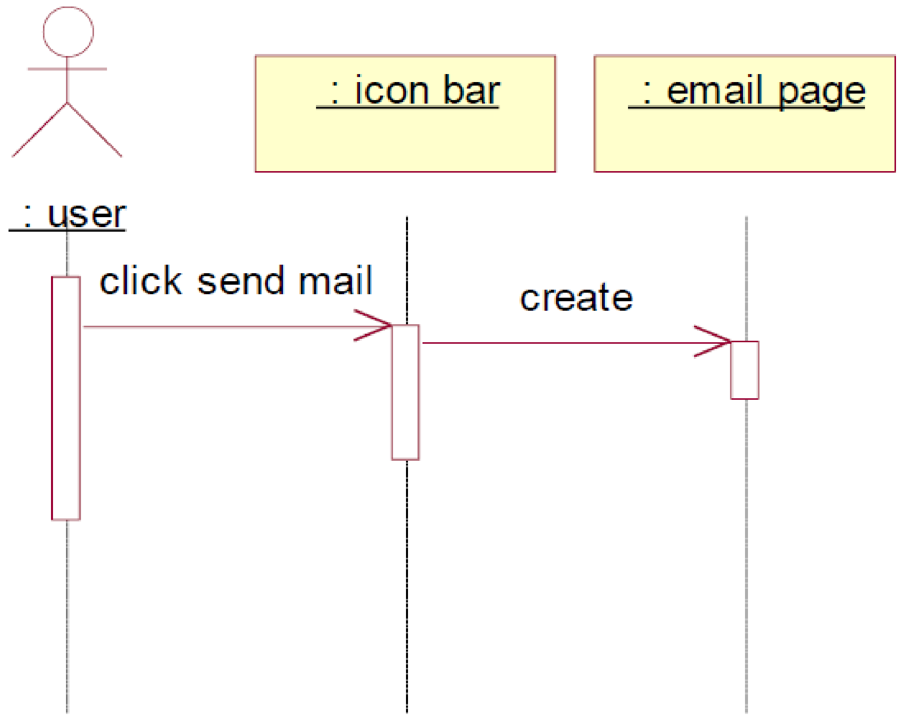
Label The Communication

- Remember that actors can only communicate with interface objects such as screens, menus and icon bars.



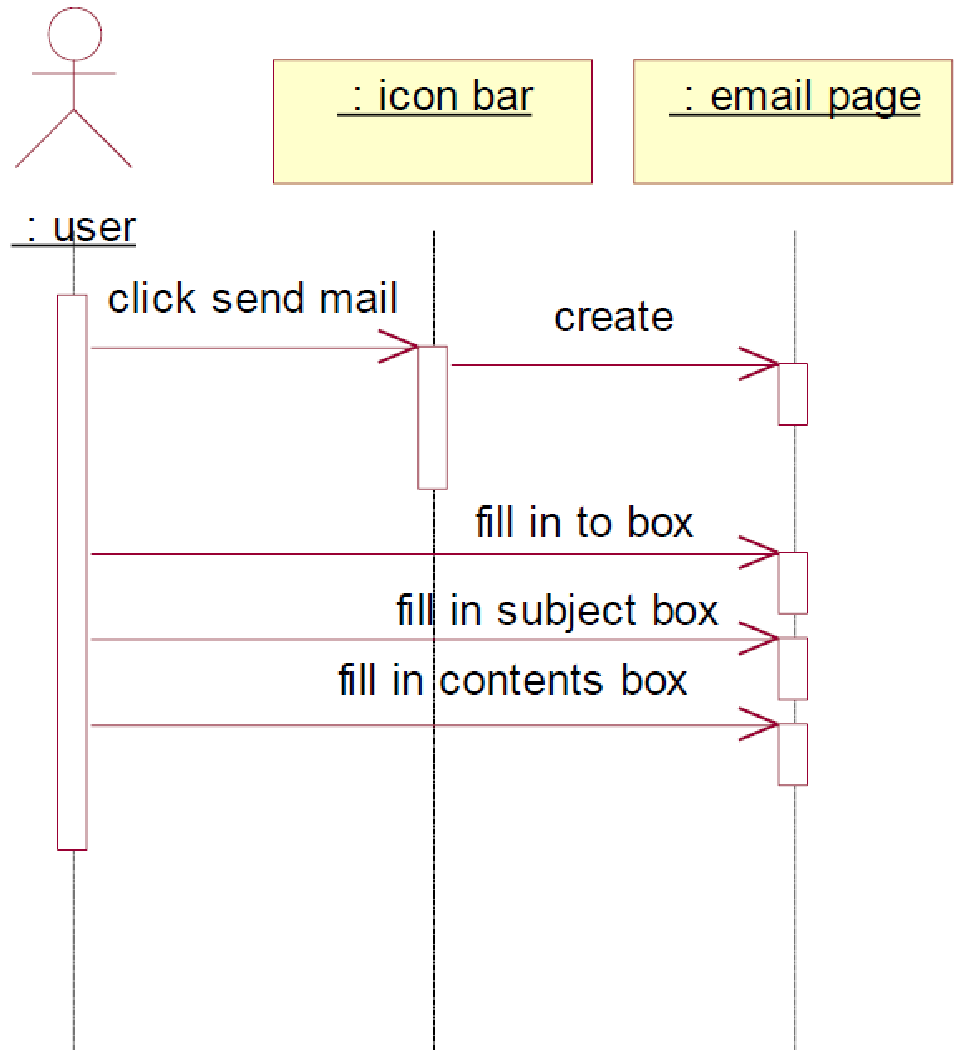
The icon bar has some work to do.

- It creates an email page.

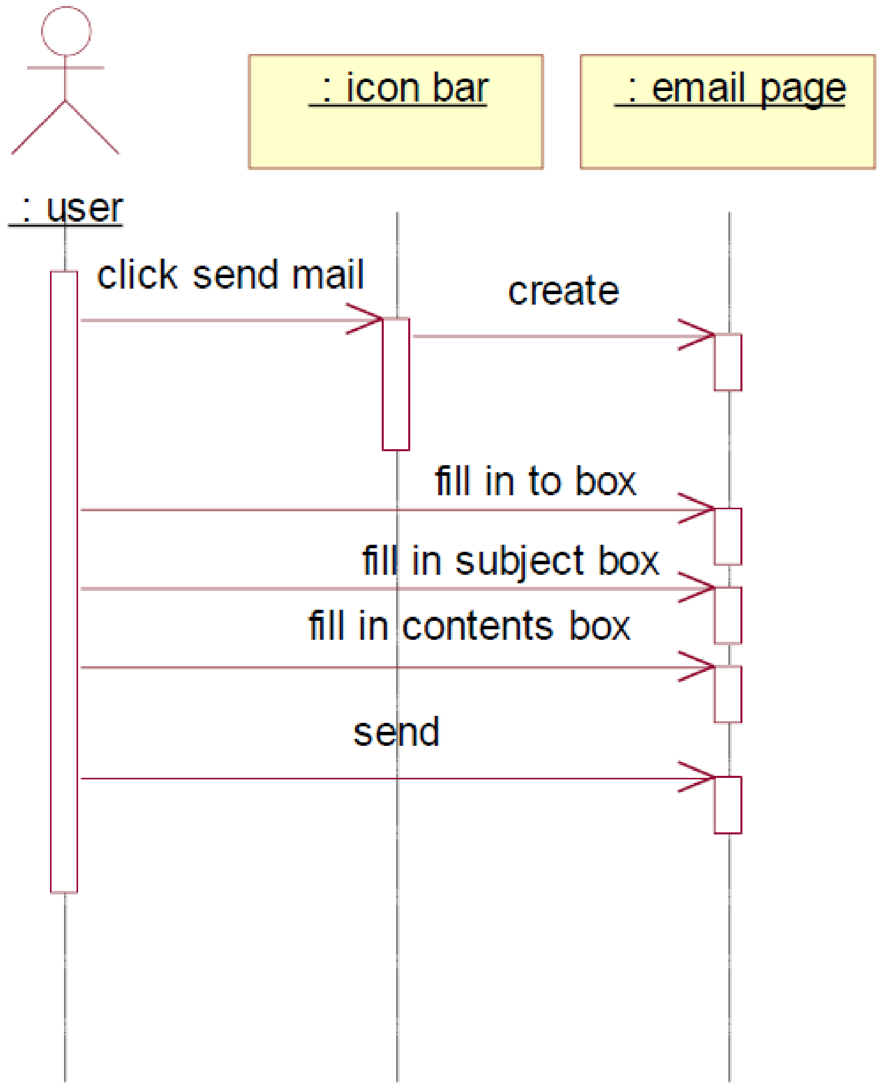


- Now the user can see the email page and use it.

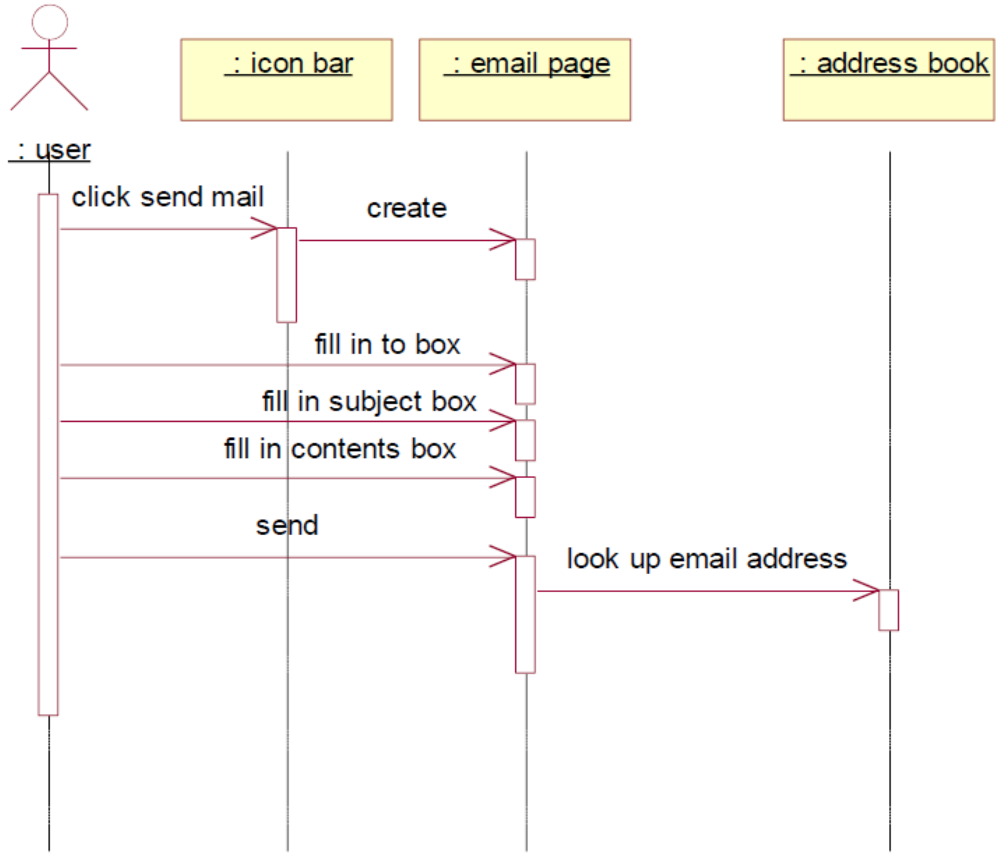
The next three steps are filling in the details on the email page



The User then clicks Send

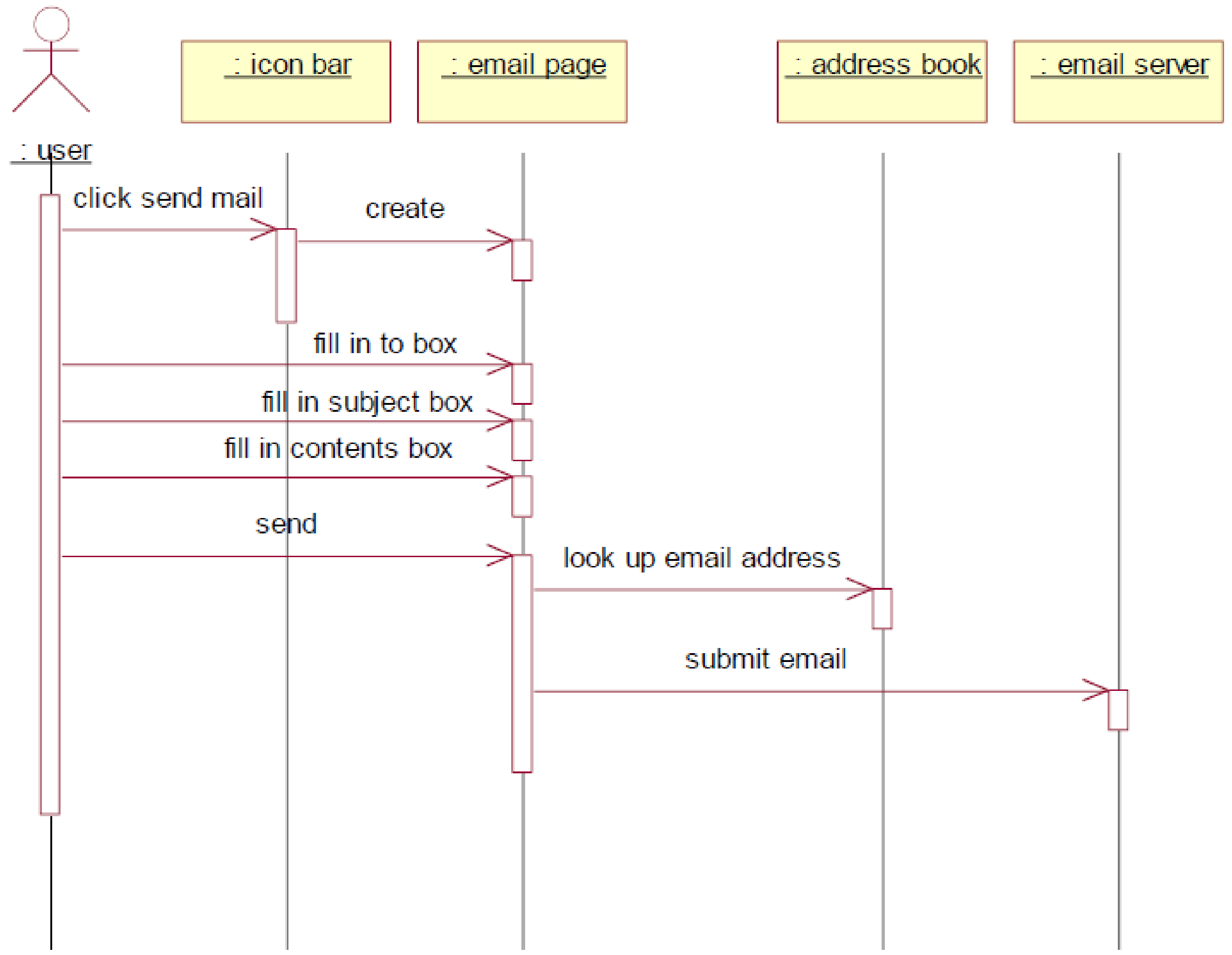


Now consider how to do the sending

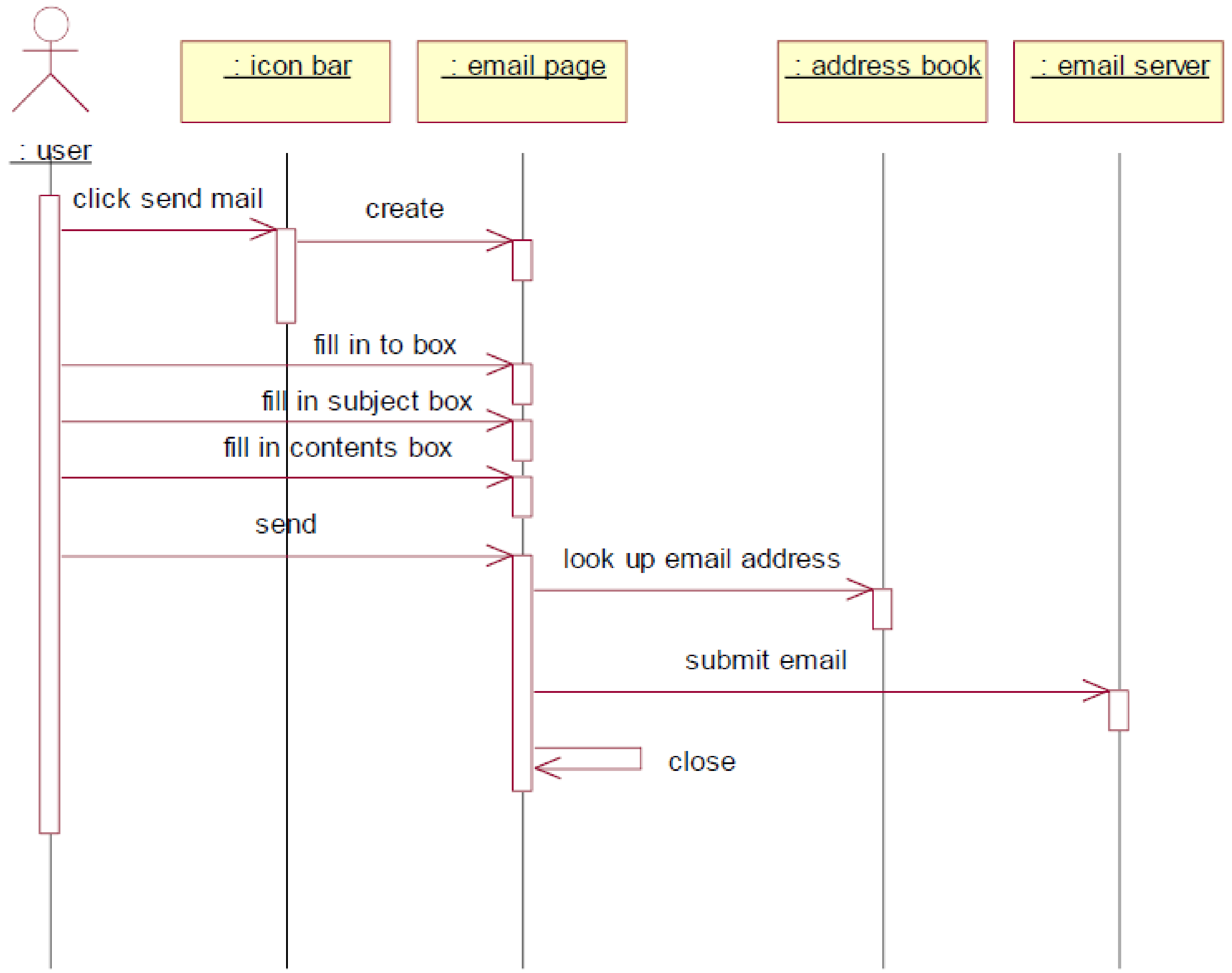


- We can choose to get the email page to look up the email address from an address book object

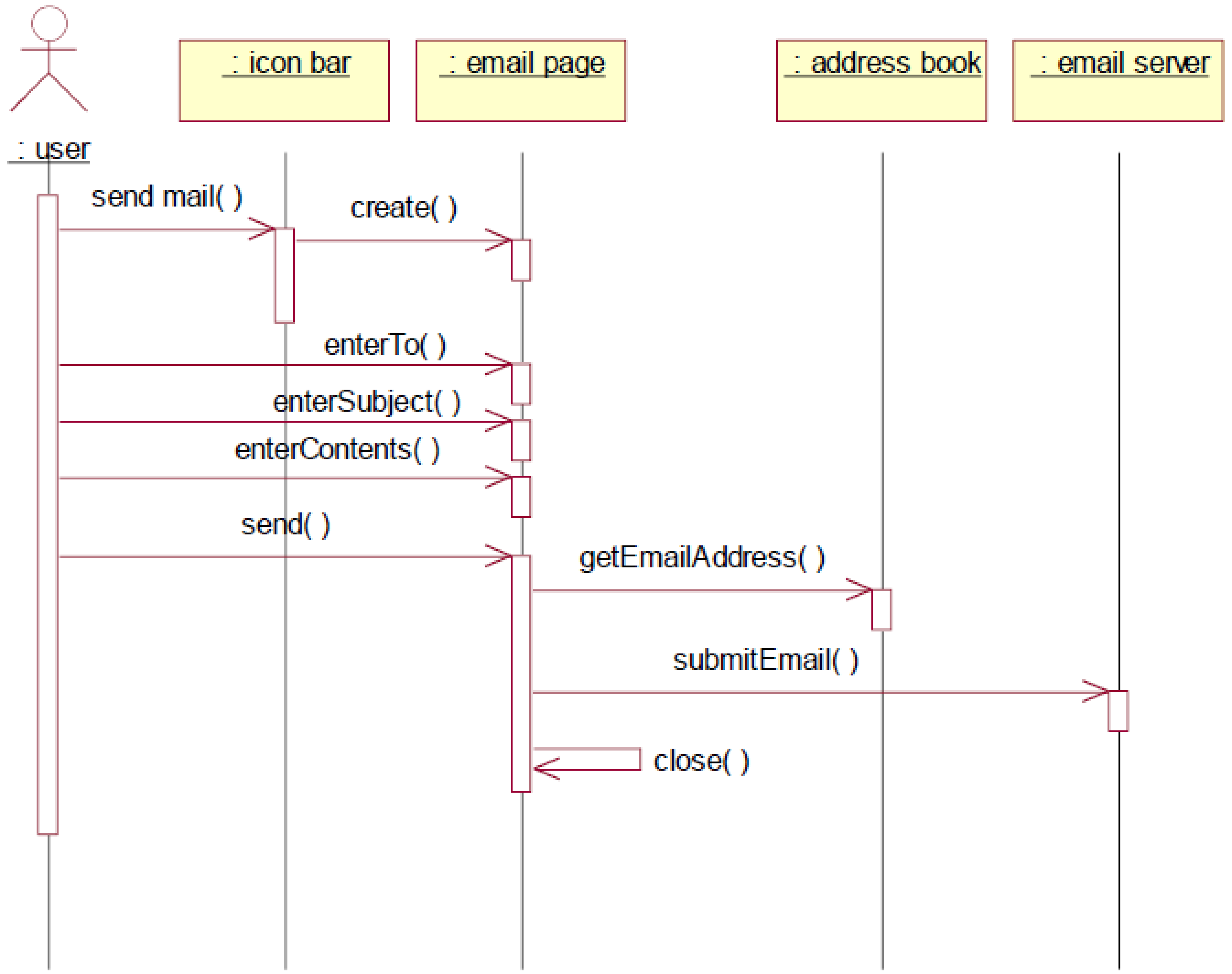
We can choose to get the email page to submit the email to the email server



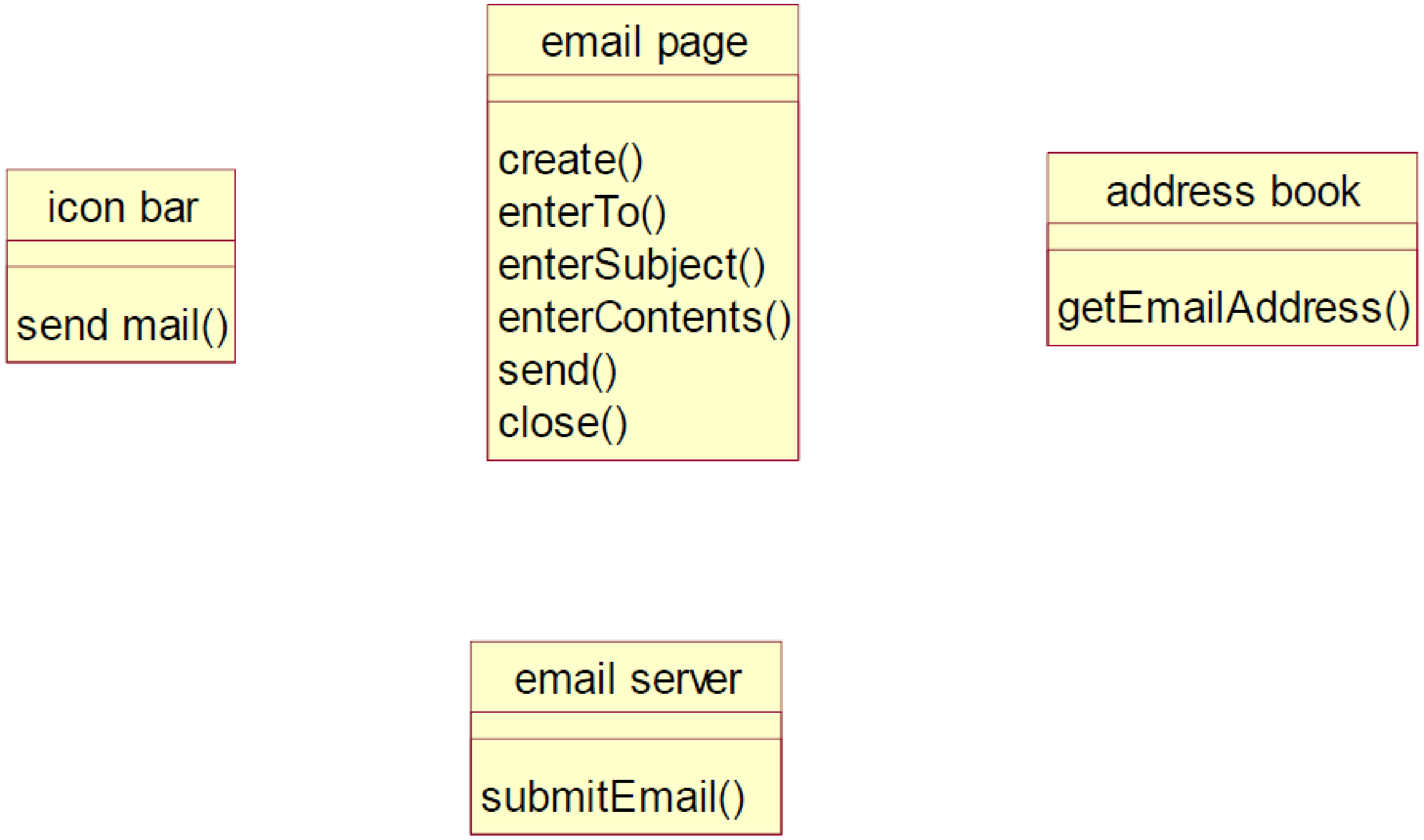
And if we think carefully, the email page always closes after the send.



Now we go through and change the messages to operations on the object

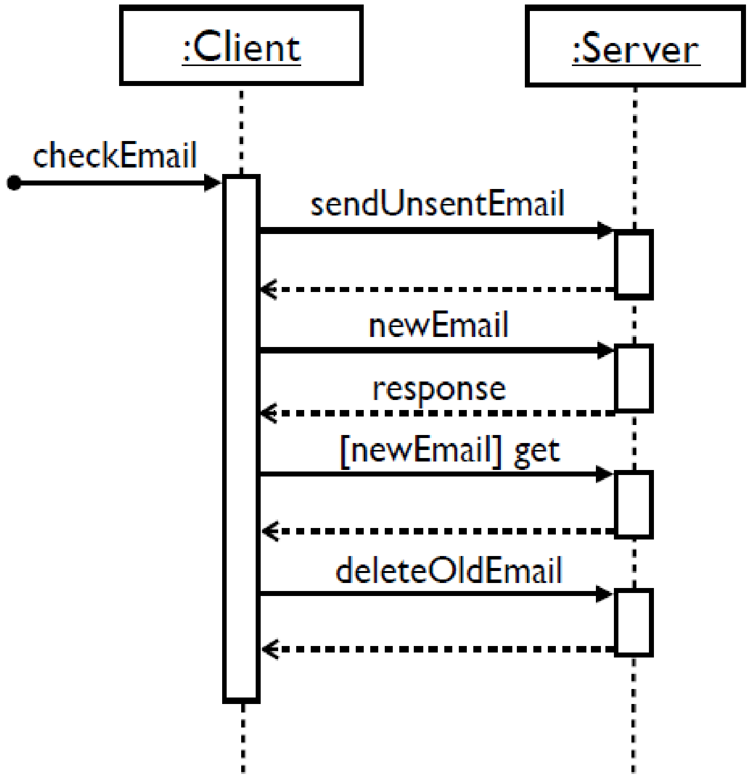


Drag them onto a class diagram



Key parts of a sequence diagram

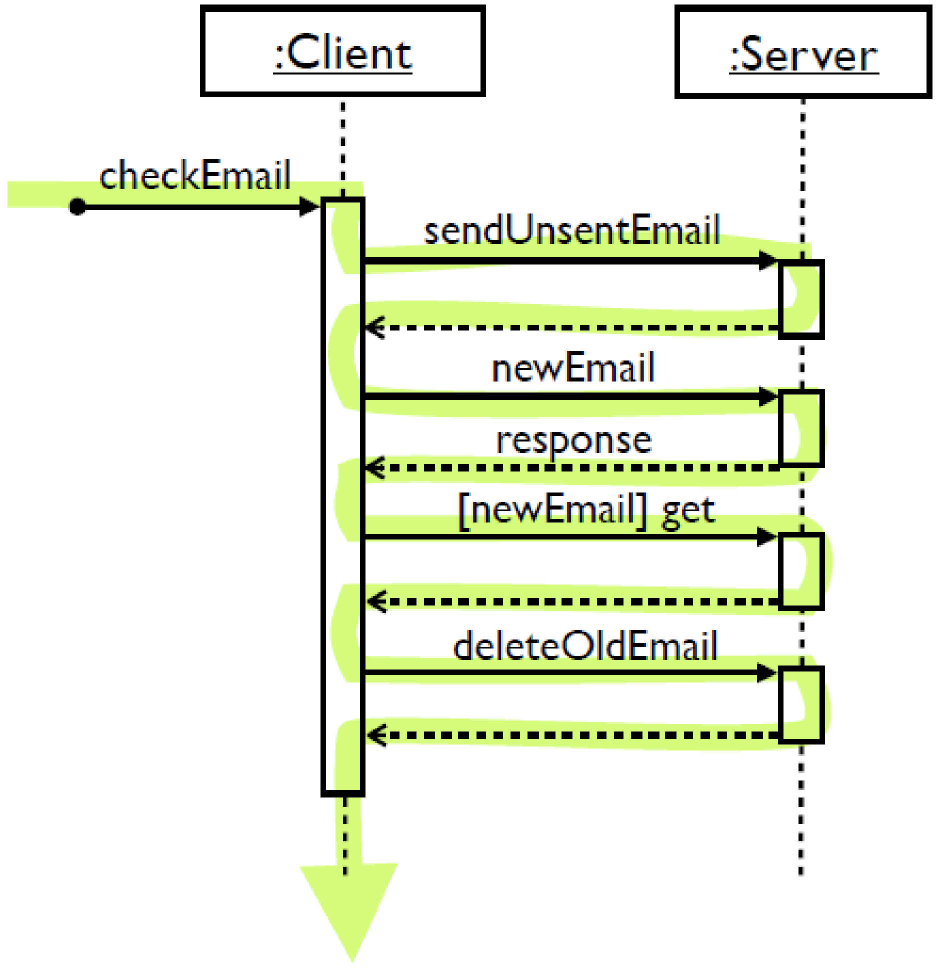
- **Participant:** an object or an entity; the sequence diagram actor sequence diagram starts with an unattached "found message" arrow. Or can be attached with an Actor of a Use Case.
- **Message:** communication between objects.



Key parts of a sequence diagram

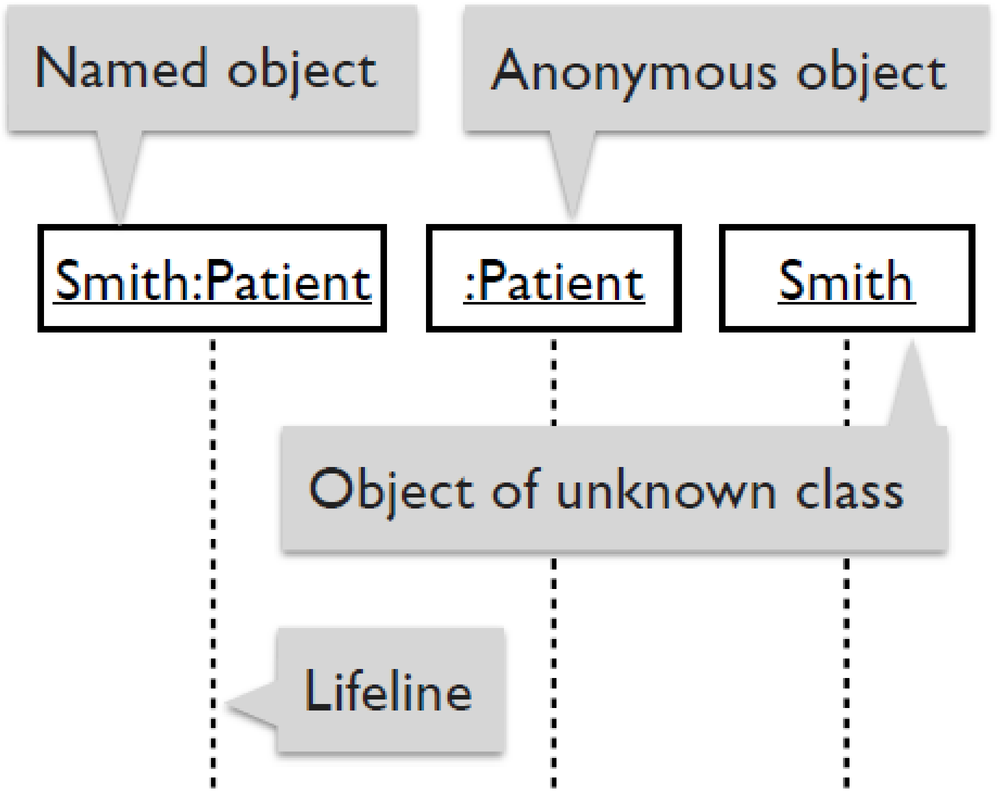
Axes in a sequence diagram:

- **horizontal:** which participant is acting
- **vertical:** time (↓ forward in time)



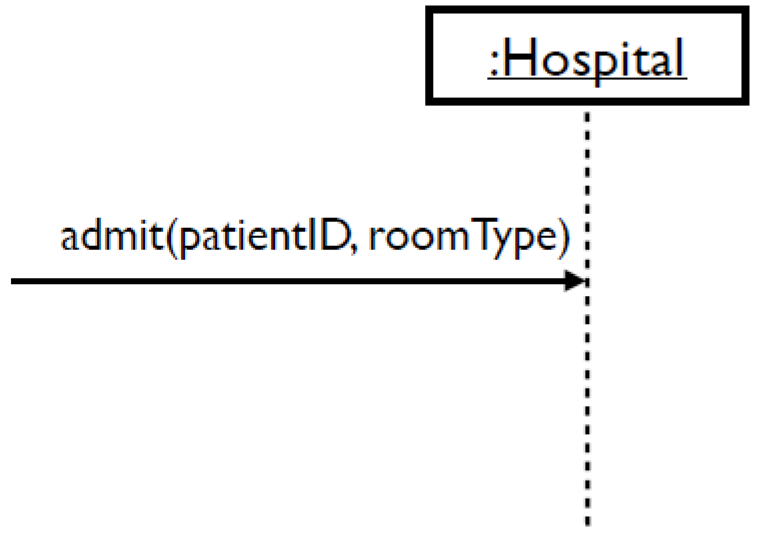
Representing objects

- objectname : classname
- An **object**: a **box** with an underlined label that specifies the object type, and optionally the object name.
  - Write the object's name if it clarifies the diagram.
- An object's "*life line*" is represented by a dashed vertical line.
  - Represents the life span of the object during the scenario being modeled.



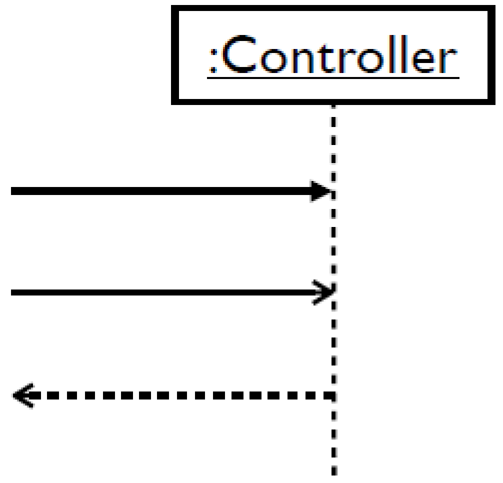
Representing messages between objects

- A **message** (method call): **horizontal arrow** to the receiving object.
  - Write message name and arguments above the arrow.



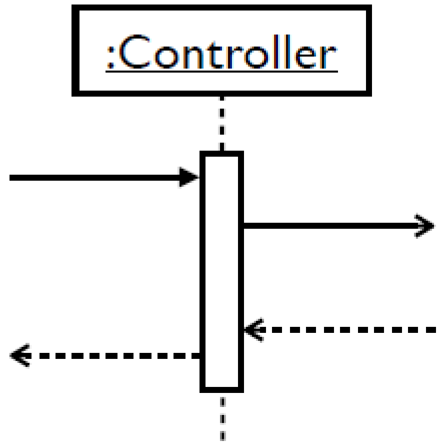
Different types of messages

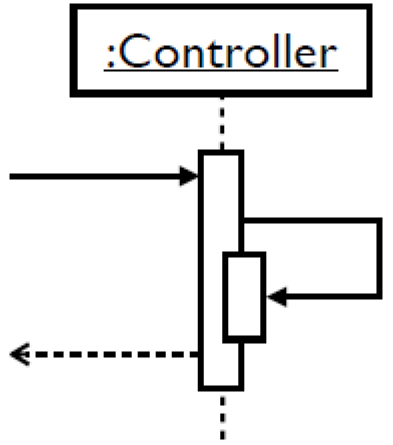
- Type of arrow indicates types of messages:
    - Synchronous message: solid arrow with a solid head.
    - Asynchronous message: solid arrow with a stick head.
    - Return message: dashed arrow with stick head.



Indicating method execution

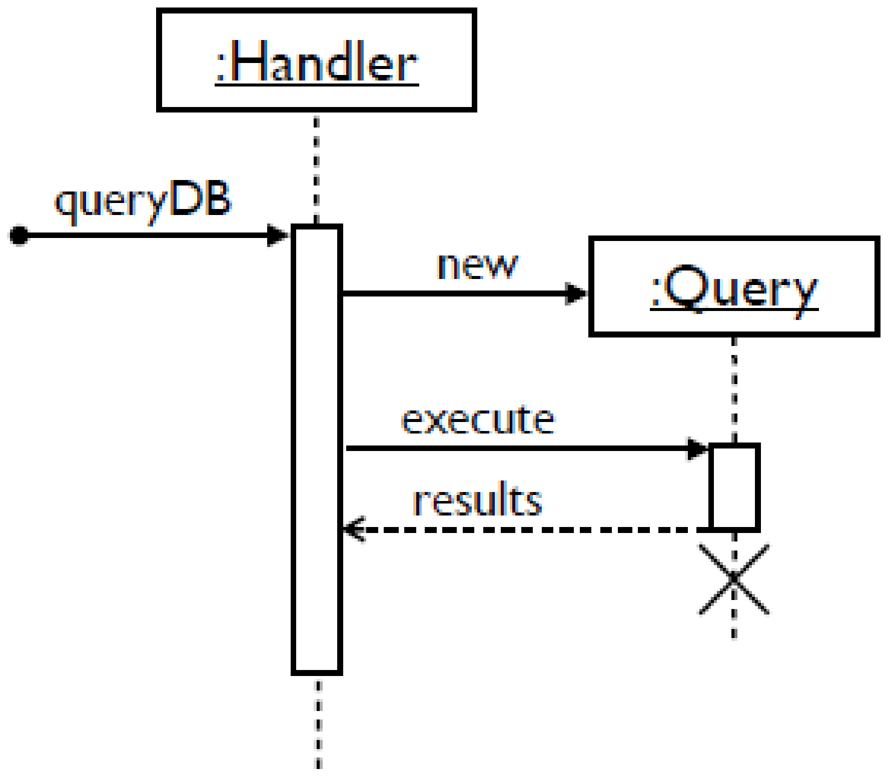
- **Activation**: thick box over object's life line, drawn when an object's method is on the stack Either that object is running its code, or it is on the stack waiting for another object’s method to finish
- Nest activations to indicate an object calling itself.





Lifetime of objects

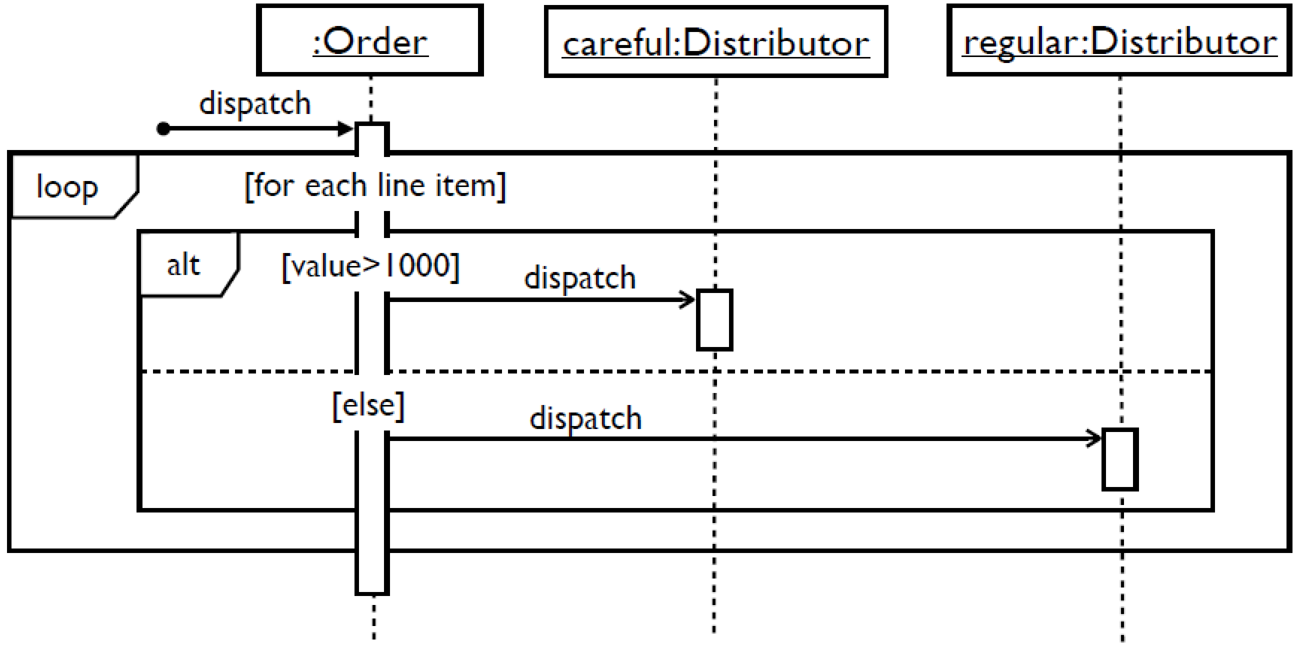
- Object **creation**: an arrow with **new** written above it
- Object d**eletion**: **X** at the bottom of object's lifeline



Alternatives, options, and loops

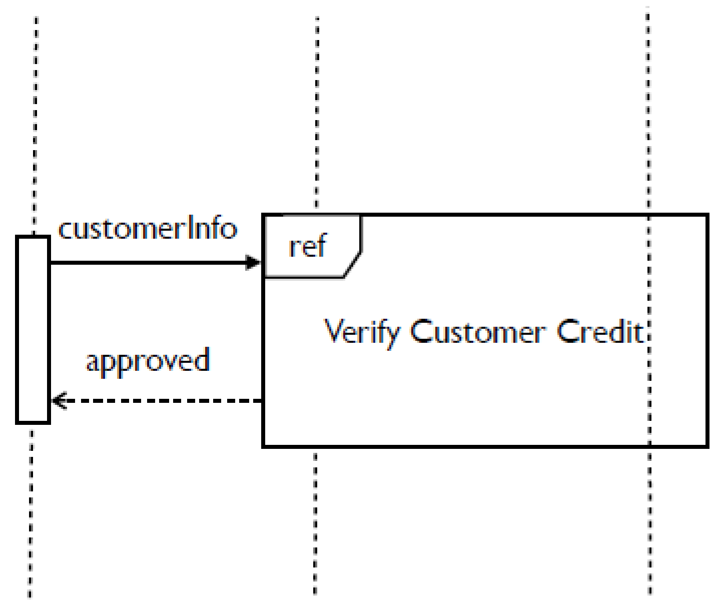
Frame: a box around part of a sequence diagram

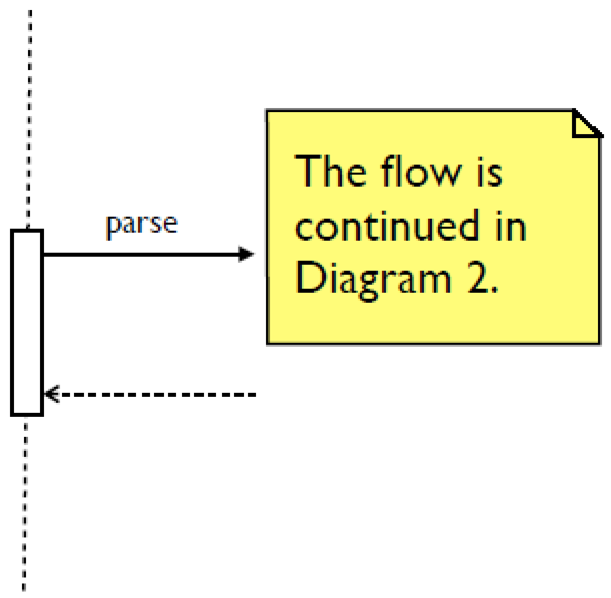
- if → (opt) [condition]
- if/else→ (alt) [condition], separated by horizontal dashed line
- loop → (loop) [condition or items to loop over]



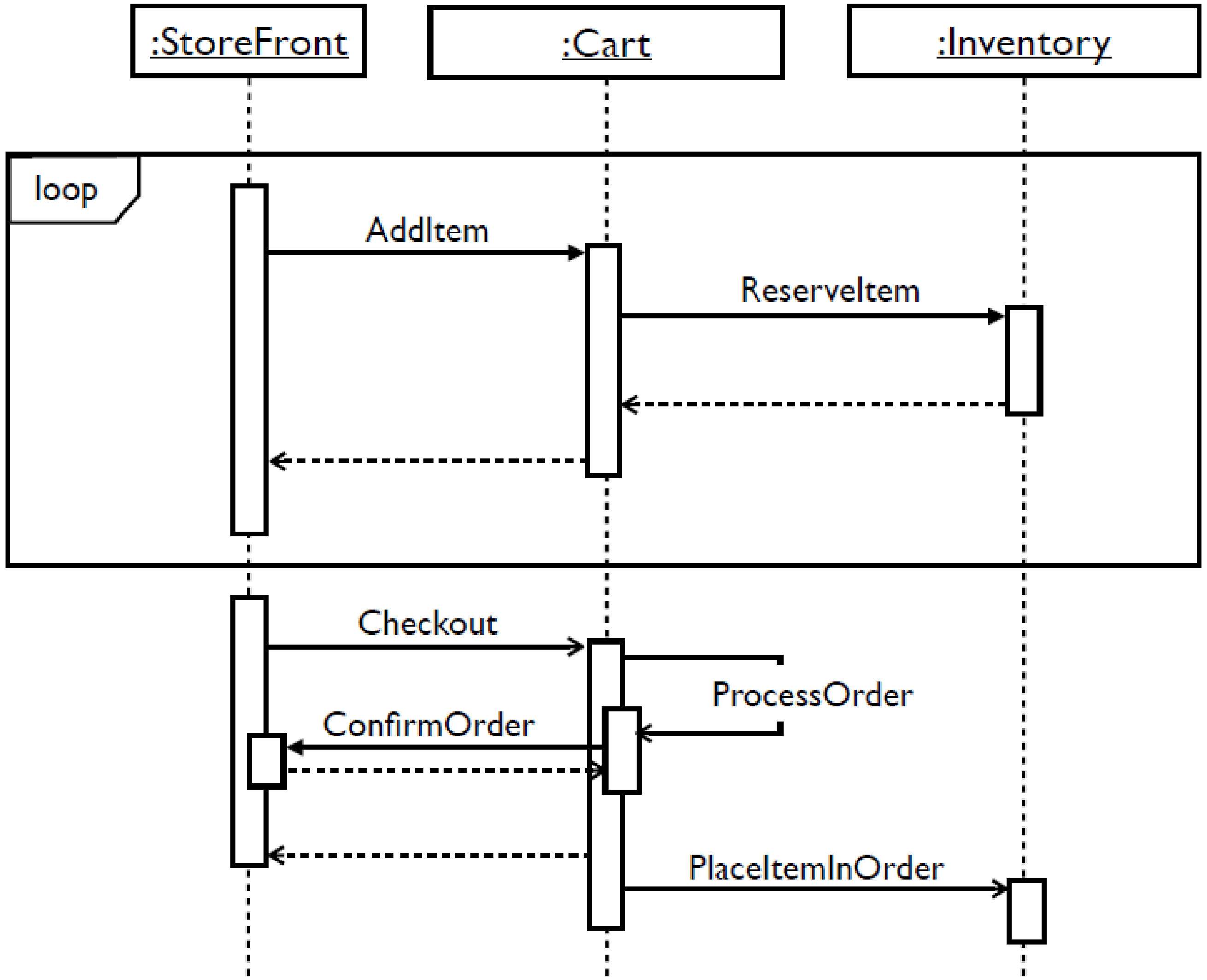
Linking sequence diagrams

- If one sequence diagram is too large or refers to another diagram:
  - An unfinished arrow and comment.
  - A **ref** frame that names the other diagram.





Example sequence diagram



Why use sequence diagrams? Why not code it?

- A good sequence diagram is still above the level of the real code (not all code is drawn on diagram)
- Sequence diagrams are language-agnostic (can be implemented in many different languages)
- Non-coders can read and write sequence diagrams.
- Easier to do sequence diagrams as a team.
- Can see many objects/classes at a time on same page (visual bandwidth).

END

☺