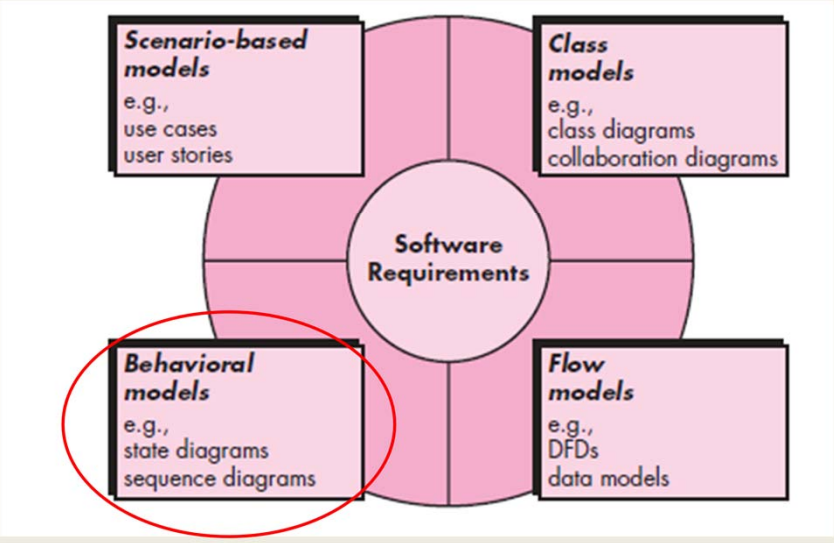
Requirements Modeling: Flow, behavior, and pattern (II)

Requirement modeling



Elements of the analysis model

Scenario-based elements

Functional—processing narratives for software functions
Use-case—descriptions of the interaction between an "actor" and the system. E.g. UML use-case diagram.

Class-based elements, E.g., UML class diagram.

Implied by scenarios.
Generalize classes, inheritance, and associations.

Flow-oriented elements

Data flow diagram

Behavioral elements

State diagram. E.g.: UML statechart.

Behavioral modeling

- What is a behavioral model?
  - System response to external events
- Why do we need behavioral modeling?
  - Application
  - Pattern
  - Web/mobile apps
- How can we create a behavioral model?
  - Events
  - State representations

Aren't scenario-based and class-based modeling representations (**static** elements of the requirements model) enough? That depends

- Behavioral modeling is often required

In some situations, complex application requirements may demand an examination of how an application behave as a consequence of external events

- Pattern is often used

Whether existing domain knowledge can be adopted for the current problem?

- Functionality of web/mobile apps

In the case of web-based or mobile systems and applications, how content and functionality meld to provide an end user with the ability to successfully navigate an application to achieve usage goals?

Behavioral modeling

- Identify events
  - understand system interaction: examining use cases

The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.

    - Underlined portions of the use case: events
      - E.g., password entered, password compared
  - Impact on control flow
    - Explicit vs. implicit
      - Examples?

To create the model, the analyst must perform the following steps: (1) Evaluate all use-cases to fully understand the sequence of interaction within the system.

-The use case represents a sequence of activities that involves actors and the system
-In general, an event occurs whenever the system and an actor exchange information
-An event is not the information that has been exchanged, but rather the fact that information has been exchanged

Homeowner transmits an event to the object **ControlPanel** The event might be called *password entered*
The information transferred is the four digits that constitute the password, but this is not an essential part of the behavioral model.

Impact on the flow of control
Some events have an explicit impact, while others have no direct impact on the flow of control
E.g.) password entered vs. password compared
The event password compared will have an explicit impact on the

information and control flow of the SafeHomesoftware

Behavioral modeling

- Connect events to objects
  - Objects generating events
  - Objects recognizing events
    - which have occurred elsewhere

The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.

Which objects generate/recognize which events?

Once all events have been identified, they are allocated to the objects involved.
Objects can be responsible for
Generating events
•(e.g., **Homeowner** generates the *password entered* event)
Recognizing events that have occurred elsewhere
•(e.g., **ControlPanel** recognizes the binary result of the *password compared* event)

Behavioral modeling

- State representations
  - which states to consider?
    - State of each class
      - system function
    - State of the system
      - external observation

state—a set of observable circumstances that characterizes the behavior of a system at a given time

In the context of behavioral modeling, two different characterizations of states must be considered:
(1) the state of each class as the system performs its function and
(2) the state of the system as observed from the outside as the system performs its function

Behavioral modeling

- State of analysis class
  - Passive vs. active state
    - Examples?
- Elements
  - State
  - State transition
  - Event
  - Action
- Representations
  - State diagram
  - Sequence diagram

The state of a class takes on both passive and active characteristics [CHA93].

A *passive state*is simply the current status of all of an object's attributes.

•E.g.) the class Player: the current position & orientation attributes

The *active state* of an object indicates the current status of the object as it undergoes a continuing transformation or processing.

•E.g.) the class Player: moving, at rest, injured, being cured, trapped, lost

state—a set of observable circumstances that characterizes the behavior of a system at a given time

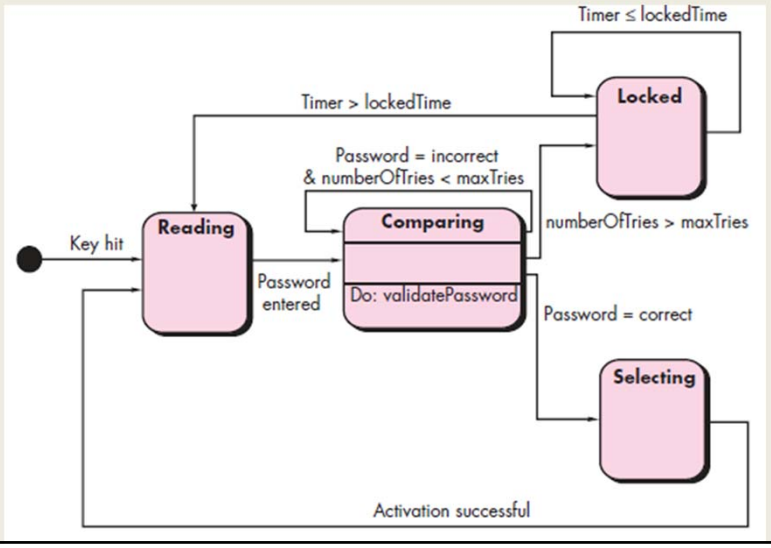
state transition—the movement from one state to another

event—an occurrence that causes the system to exhibit some predictable form of behavior

action—process that occurs as a consequence of making a transition

Representation: state diagram

- Class/object's active states and transitions among them
  - Example – the *ControlPanel* class



A UML state diagram

Represents active states for each class and the events (triggers) that cause changes between these active states.

Notations

State transition: Each *arrow* represents a transition from one active state of an object to another.

Event: The *labels* shown for each arrow represent the event

Condition for transition: A *guard* is a Boolean condition that must be satisfied in order for the transition to occur.

•E.g.) if (password input = 4 digits) then compare to stored password

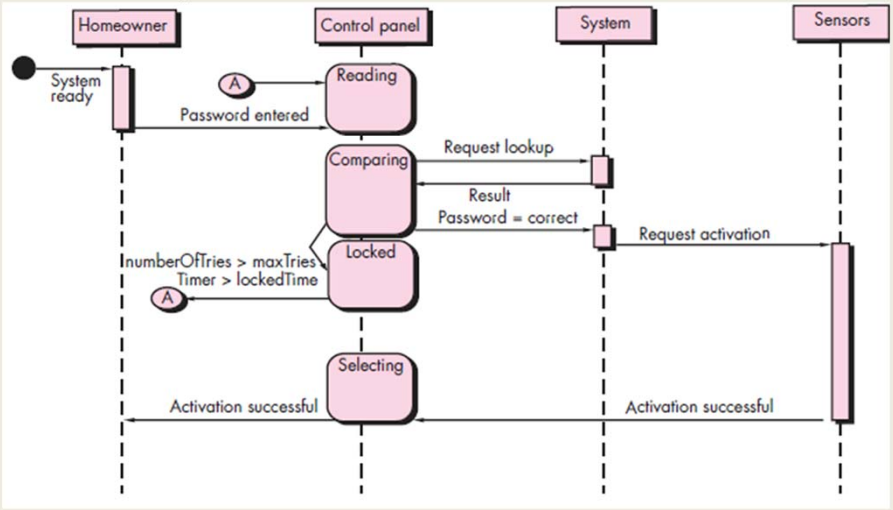
•The guard depends on the passive state of the object

Action: An *action* occurs concurrently with the state transition or as a consequence of it and generally involves one or more operations (responsibilities) of the object.

•E.g.) an operation named *validatePassword()* performs a digit-by-digit comparison to validate the entered password.

Representation: sequence diagram

- Event-object interaction over time
  - One SD per use case



sequence diagram in UML, indicates how events cause transitions from object to object **as a function of time**.
sequence diagram is used to show the dynamic communications between objects during execution of a task. It shows the temporal order
in which messages are sent between the objects to accomplish that task. One might use a sequence diagram to show the interactions in one use case or in one scenario of a software system.

A sequence diagram shows step by step what must happen to accomplish a piece of functionality provided by the system.
Shows object interactions arranged in time sequence. In particular, it shows the objects participating in an interaction and the sequence of messages exchanged.

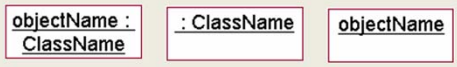
A SD captures the behavior of a single scenario. There will be a SD for each use case in the system.

We draw SDs in order to document how objects collaborate to produce the functionality of each use case.

The SDs show the data and messages which pass across the system boundary and the messages being sent from one object to another in order to achieve the overall functionality of the use case.

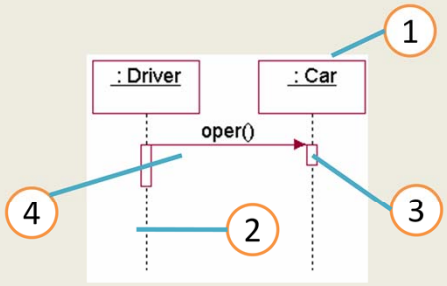
Sequence diagram: notations

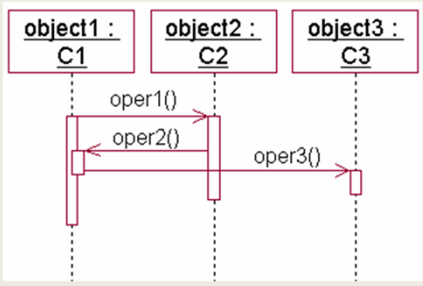
- 1. Object (w/ or wo/ class)



- 2. lifeline
  - Possibly ending with an 'X'
- 3. activation bar
- 4. method call (message)

Activation can be nested





1. Objects: appear at the top of the column

Different with "classes"

2. Lifeline: A dashed line that shows the existence of an object over a period of time.

Put a "X" at the end to indicate destruction. (optional)

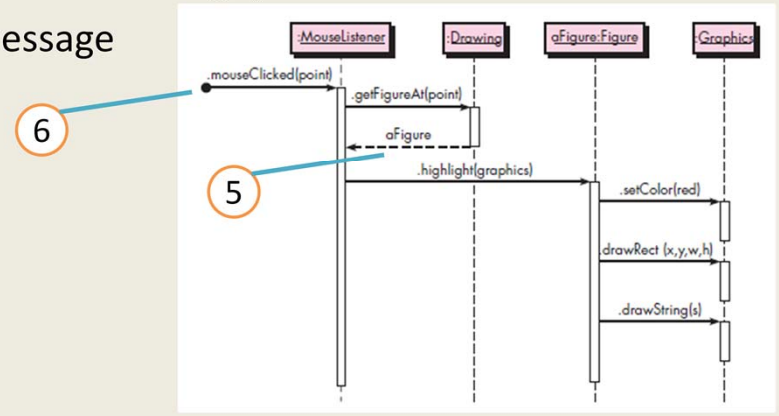
3. A symbol that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.

When an object is executing a method (that is, when it has an activation frame on the stack), you can optionally display a white bar, called an *activation bar*, down the object's lifeline.

4. A sequence diagram shows method calls using horizontal arrows from the *caller* to the *callee*, labeled with the method name and optionally including its parameters, their types, and the return type.

Sequence diagram: notations

- 5. return (return message)
- 6. found message



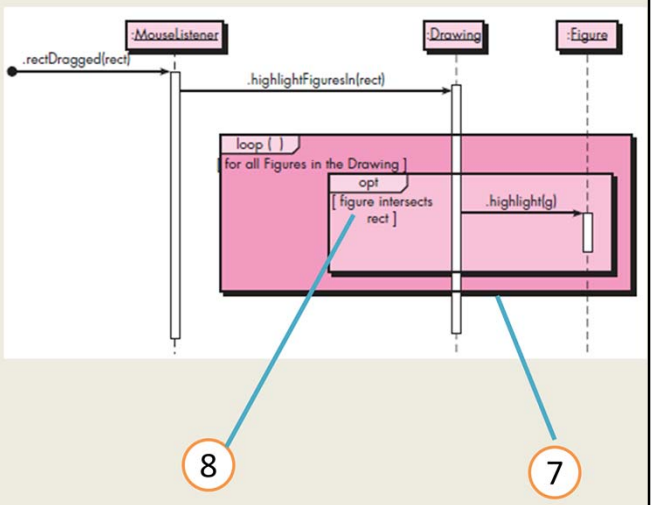
5. The diagram can also optionally show the return from a method call with a dashed arrow and an optional label with the name of the object that was returned

-- A common practice is to leave off the return arrow when a void method has been called, since it clutters up the diagram while providing little information of importance

6. A black circle with an arrow coming from it indicates a *found message* whose source is unknown or irrelevant.

Sequence diagram: notations

- 7. interaction frame
- 8. guards



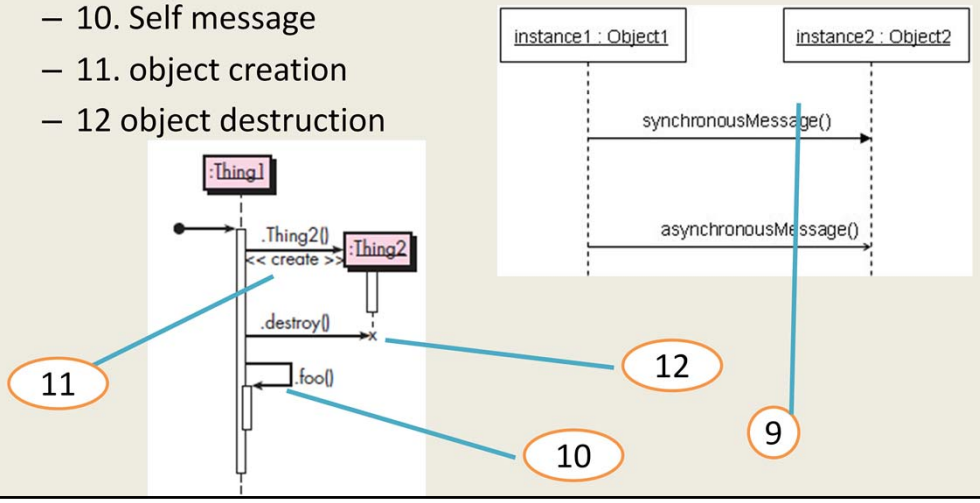
*** If logical control structures are required, it is probably best to draw a separate sequence diagram for each case. That is, if the message flow can take two different paths depending on a condition, then draw two separate sequence diagrams, one for each possibility.

7. If you insist on including loops, conditionals, and other control structures in a sequence diagram, you can use *interaction frames*, which are rectangles that surround parts of the diagram and that are labeled with the type of control structures they represent.

8. The phrases in square brackets are called *guards*, which are Boolean conditions that must be true if the action inside the interaction frame is to continue.

Sequence diagram: notations

- More
  - 9. Synchronous vs. asynchronous message
  - 10. Self message
  - 11. object creation
  - 12 object destruction



9.Synchronous mess ages are shown with solid arrowheads while asynchronous messages are shown with stick arrowheads.

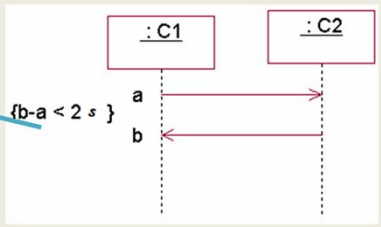
10.You can show an object sending itself a message with an arrow going out from the object, turning downward, and then pointing back to the same object.

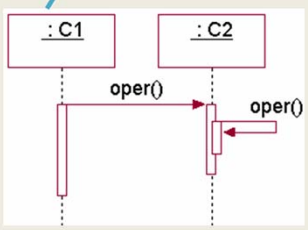
11.You can show object creation by drawing an arrow appropriately labeled (for example, with a «create» label) to an object's box. In this case, the box will appear lower in the diagram than the boxes corresponding to objects already in existence when the action begins.

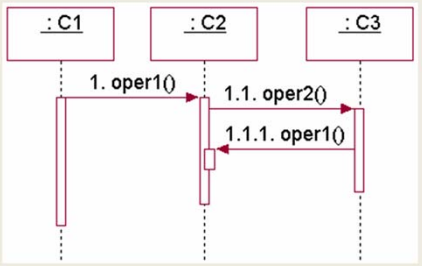
12.You can show object destruction by a big X at the end of the object's lifeline. Other objects can destroy an object, in which case an arrow points from the other object to the X.

Sequence diagram: notations

- More
  - Time constraint
  - Recursive calls



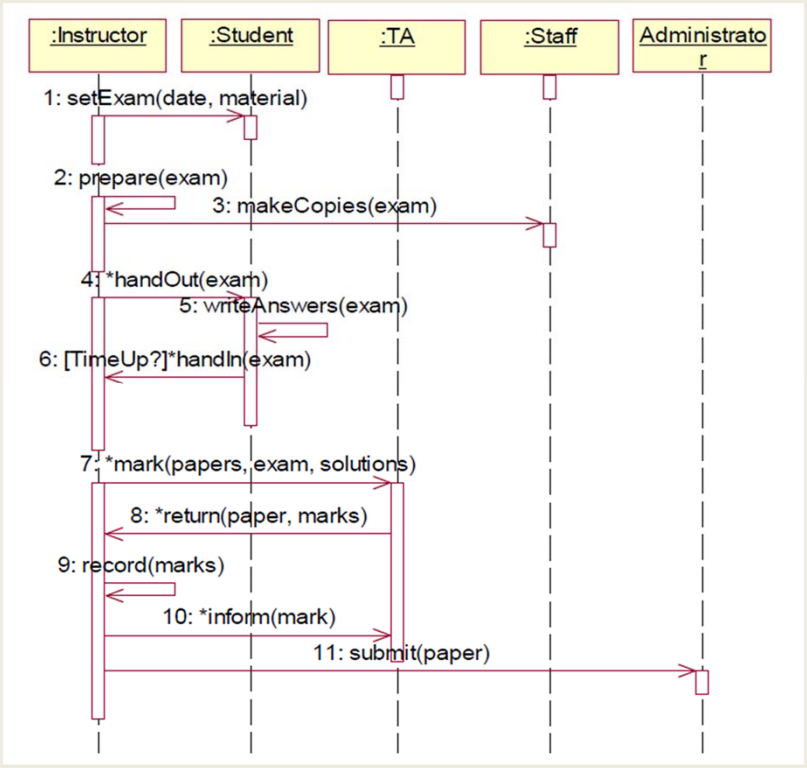




Exercise

- The interaction of an exam!

The instructor first informs the students the time, location and content of the exam. Then he prepares the exam and the solution, and print out enough copies, to be distributed at the appointed time and location. The students take and return the exams. The instructor then give the solution to the TAs, who give him back the grades. Finally, he submit both the grades and exams to the department.


:Instructor
:Student
:TA
:Staff
Administrato
r
1: setExam(date, material)
2: prepare(exam)
3: makeCopies(exam)
4: *handOut(exam)
5: writeAnswers(exam)
6: [TimeUp?]*handIn(exam)
7: *mark(papers, exam, solutions)
8: *return(paper, marks)
9: record(marks)
10: *inform(mark)
11: submit(paper)

Summary

- Behavioral modeling
  - What is a behavioral model
  - Why do we need it
  - How do we do the modeling
- Representations
  - UML state diagram
  - UML sequence diagram