Software Design: Sequence Diagrams +State Diagrams + Midterm Info.

Content in part from Chapter 5 of "Head First Software Development", Pilone et al.

Miami University Software Technology & Analysis Group (MUSTANG)
Computer Science & Software Engineering
Miami University, Oxford, Ohio, USA

Week's Agenda

- Assignment #2 solution
- Project introduction
  - Fill out schedule survey this week.
  - Due by Friday!
  - 1st weekly customer meetings begin next week!
- Software design
  - Advanced UML
  - Sequence Diagrams & State Diagrams
- Midterm information
  - Thursday March 11th

ADMIN

- Project started
  - Deliverables for this week in Canvas
  - Deliverables for next week's meeting posted in Canvas

- Midterm Thursday March 11
  - Summarizing end of today

Questions From Last Class?

- Project Overview
- Class Diagrams
  - Association (Uni- and Bi)
  - Aggregation
  - Composition
  - Inheritance
- Exercise

UML SEQUENCE (?) DIAGRAMS (SD)

UML – Three Different Models of Design

- **Functional model** – functionality of the system
  - Use-case diagrams
- **Object model** – structure of the system
  - Class diagrams
- **Dynamic model** – behavior of the system
  - Sequence diagrams & State chart diagrams

WHY NOT GO RIGHT FROM REQS. TO CODING?

- The team developing the SRS or User Stories may not be the team developing the software!
- Different skill sets, levels
- Requirements analysts are often more experienced with capturing behavior than developers

Dynamic Modeling

- Definition:
  - Important Dynamic Behavior
    - Statechart diagram for a single class
    - Sequence chart diagram with important dynamic behavior
- Purpose:
  - Detect and supply methods for the object (structural) model
- How:
  - Start with use case scenario
  - Model interaction between objects – sequence diagram
  - Model dynamic behavior of a single object – state chart diagram.

Sequence Diagrams

- Formalize behavior of the system
- Visualize communication of objects
- Depict communication between objects during a use-case
  - Bars are used to represent the lifetime of a particular object
  - Lines are used to show method invocation
- Start with flow of events from the use case model when attempting to generate a sequence diagram

UML Sequence Diagrams

- **Sequence diagram**: an "interaction diagram" that models a single scenario executing in the system
  - One of the most (2nd) used UML diagram (behind class diagram)

- Relation of UML diagrams to other exercises:
  - CRC cards -> class diagram
    - https://en.wikipedia.org/wiki/Class-responsibility-collaboration_card
  - Use cases -> sequence diagrams

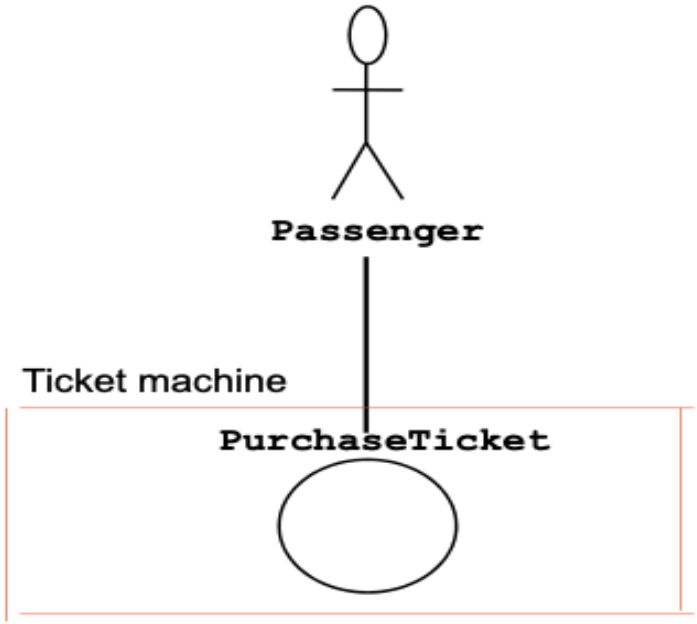
What is an Event?

- Something that happens at a point in time
- Relation of events to each other:
  - Causally related: one occurs before or after another
  - Causally unrelated: concurrent (?)
- An event sends information from one object to another – think about method invocation

Elements of a Sequence Diagram

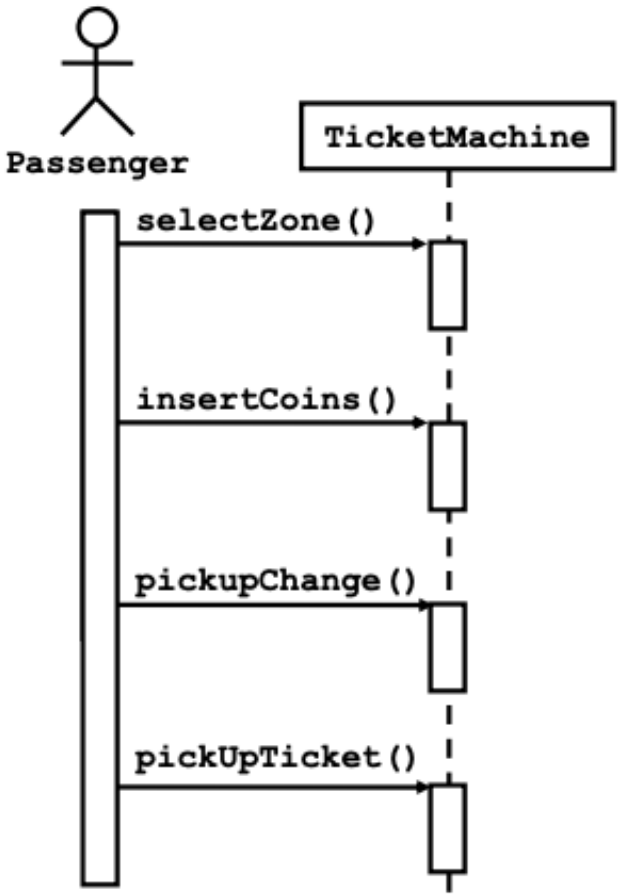
- **Classes**: are represented by columns
- **Messages**: are represented by arrows
- **Activations**: are represented by narrow rectangles
- **Lifelines**: are represented by dashed lines

USE-CASE EXAMPLE



- Name: Purchase ticket
- Event flow:
  1. Passenger selects the number of zones to be traveled.
  2. Machine displays the amount due.
  3. Passenger inserts money, of at least the amount due.
  4. Machine returns change.
  5. Machine issues ticket.
- >>>> use event flow to generate sequence diagram

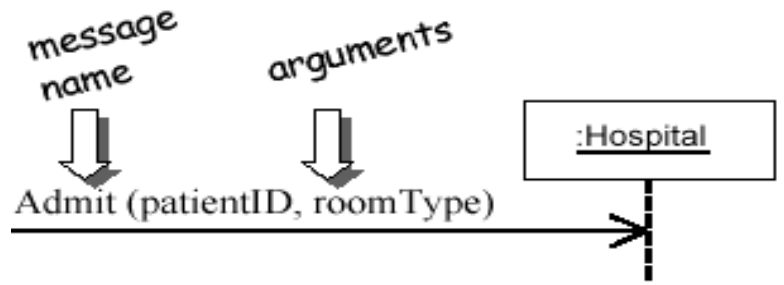
UML Sequence Diagrams



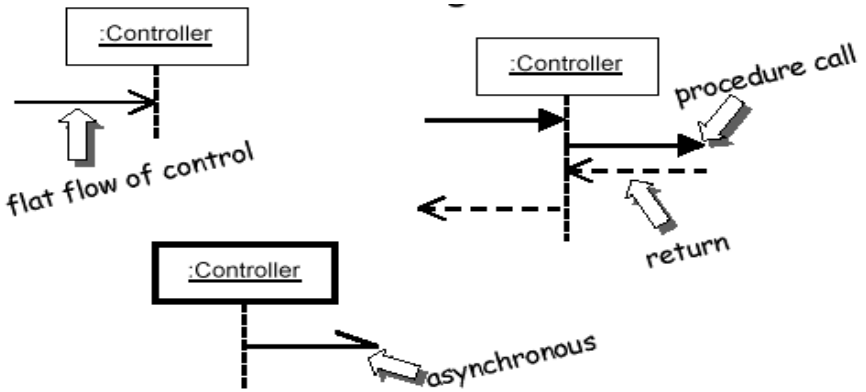
- Diagram elements revisited
  - ***Classes*** are represented by columns
  - ***Messages*** are represented by arrows
  - ***Activations*** are represented by narrow rectangles
  - ***Lifelines*** are represented by dashed lines

MESSAGES BETWEEN OBJECTS

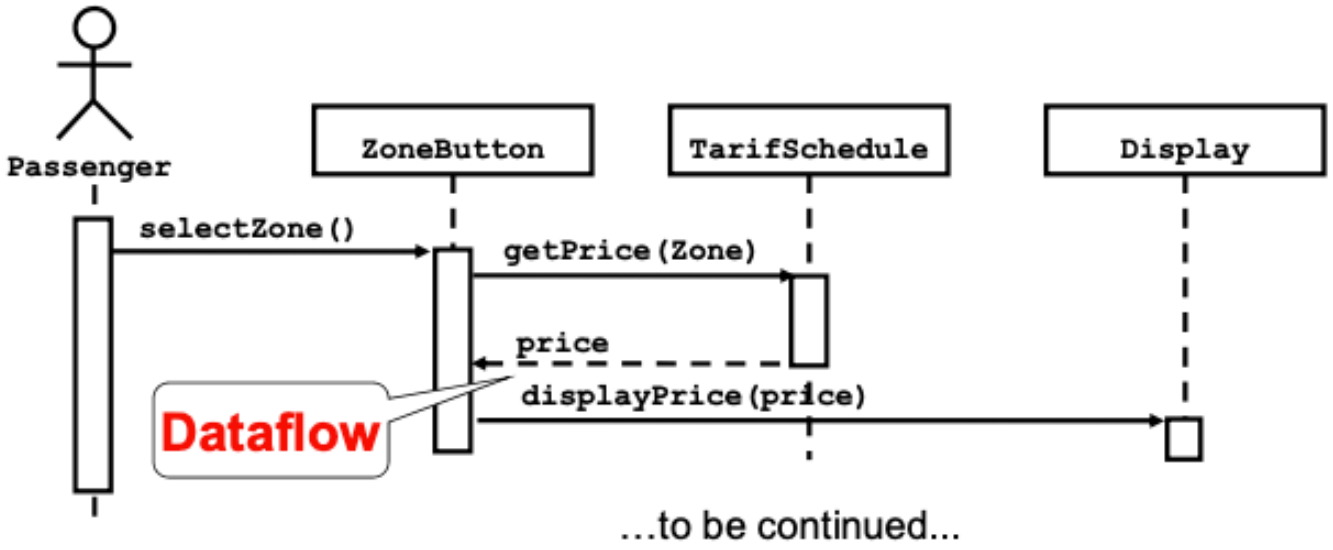
- Message (method call) indicated by horizontal arrow to other object
  - Write message name and arguments above arrow



  - Dashed arrow back indicates return
  - Different arrowheads for normal / concurrent (asynchronous) methods



UML SD: Nested Messages

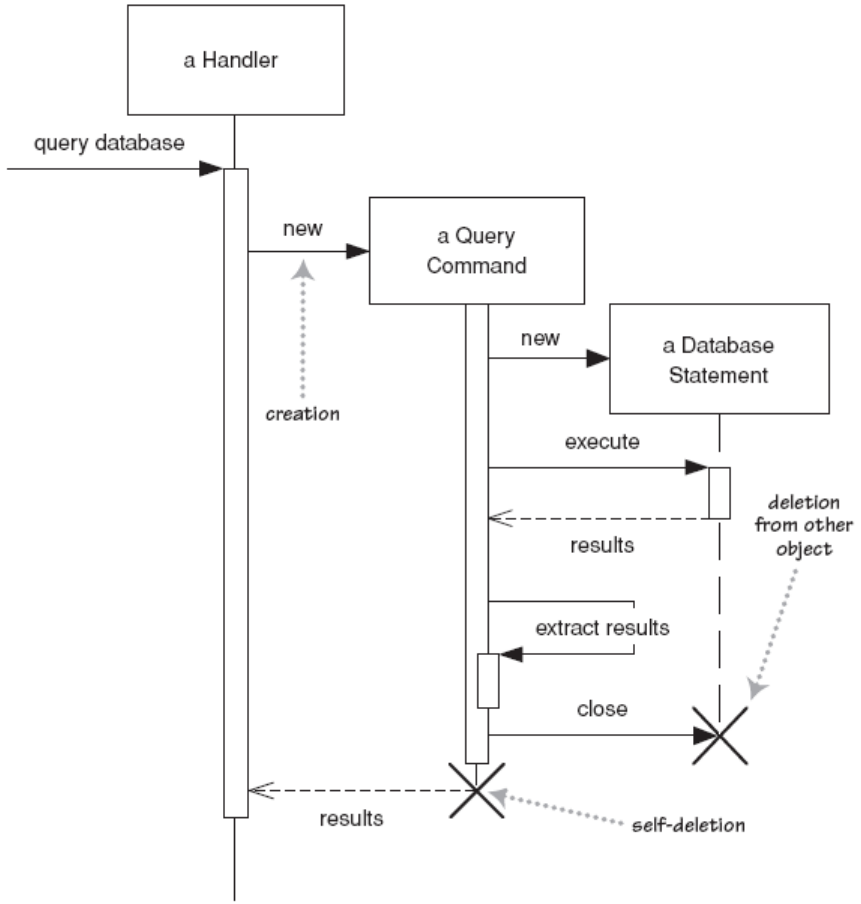


- The source of an arrow indicates the activation which sent the message
- An activation is as long as all nested activations
  - i.e. until the requirement is over

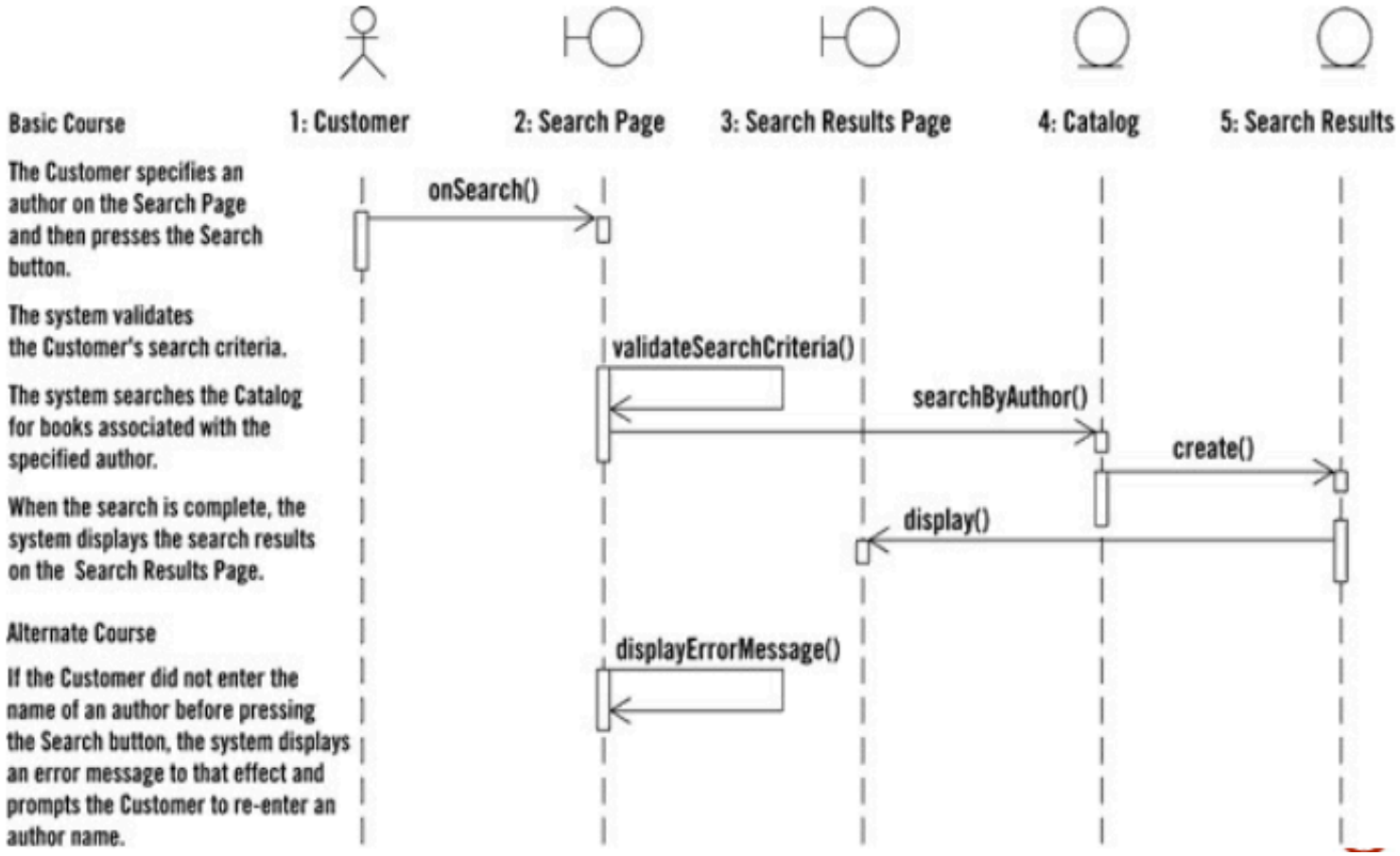
LIFETIME OF OBJECTS

- **Creation**: arrow with 'new' written above it
  - Notice that an object created after the start of the scenario appears lower than the others

- **Deletion**: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected

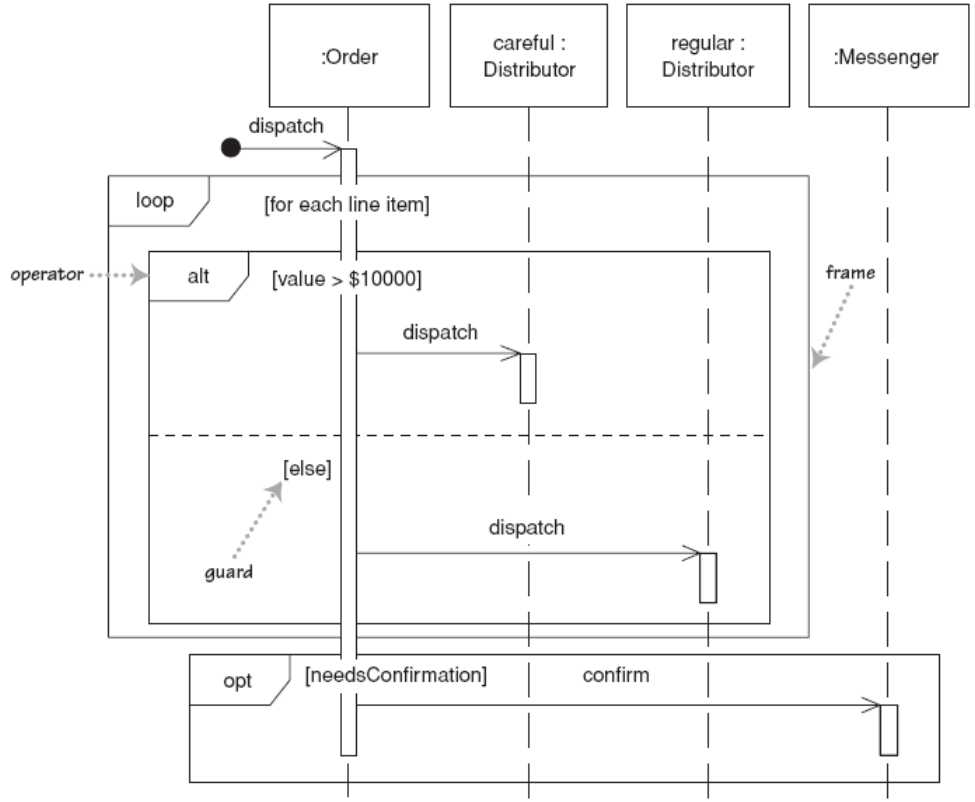


ANOTHER (ADVANCED) EXAMPLE



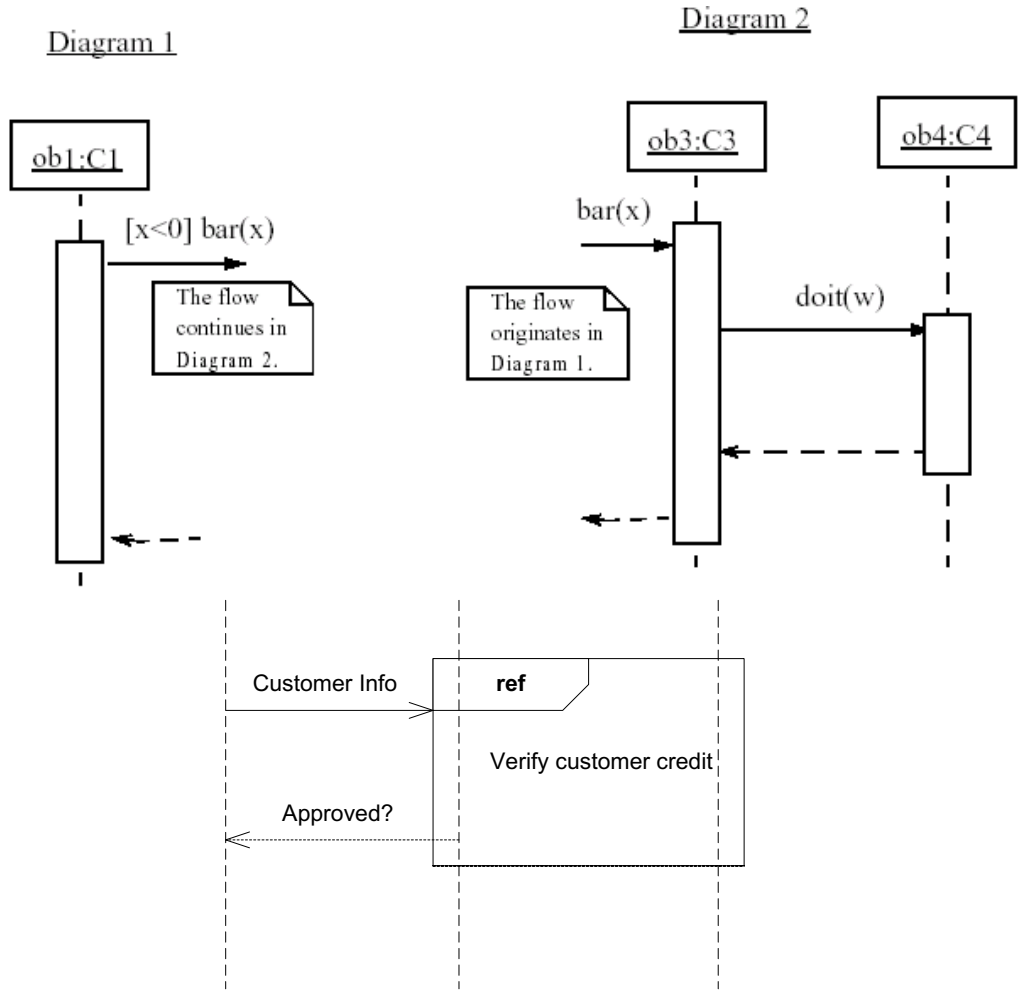
INDICATING SELECTION AND LOOPS

- **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]



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
  - When would this occur in our system?



SD OBSERVATIONS

- SD represents behavior in terms of interactions
- Complement the class diagrams, which represent structure
- Useful to find participating objects
- Time consuming but worth the investment.

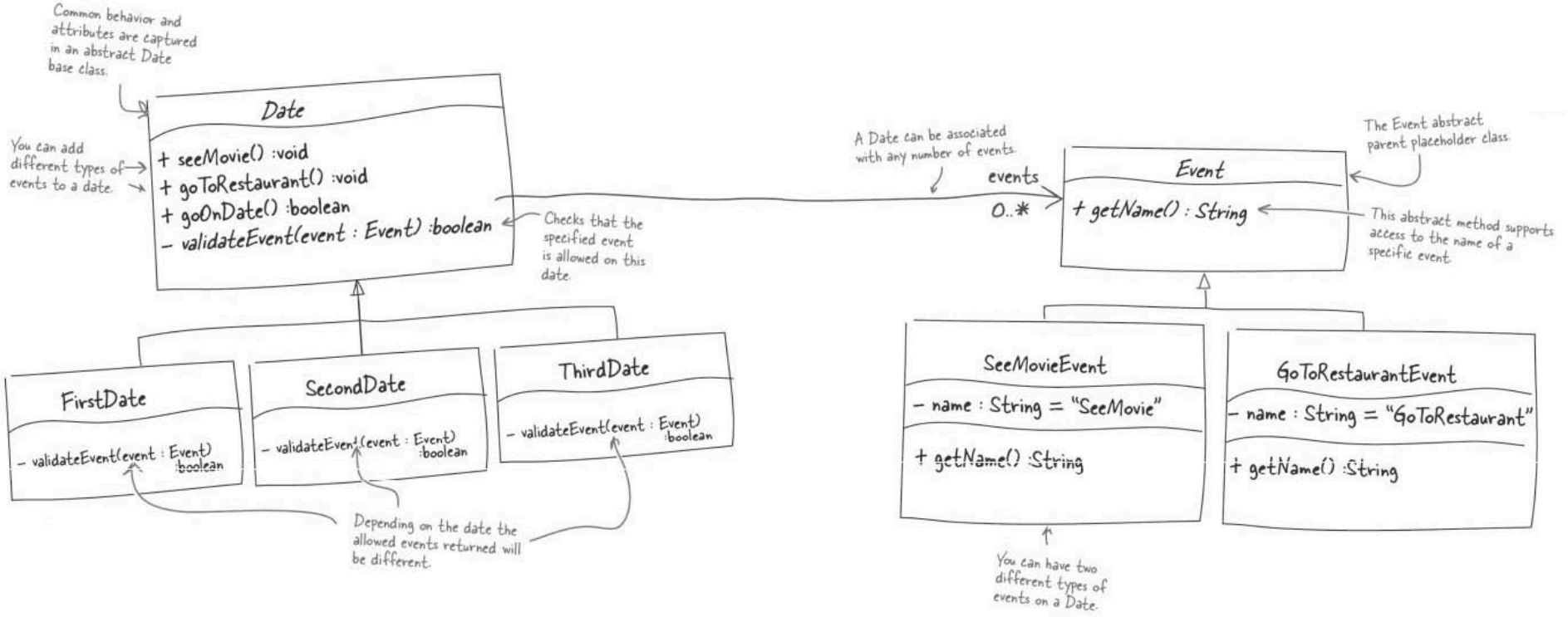
WHY NOT JUST CODE IT?

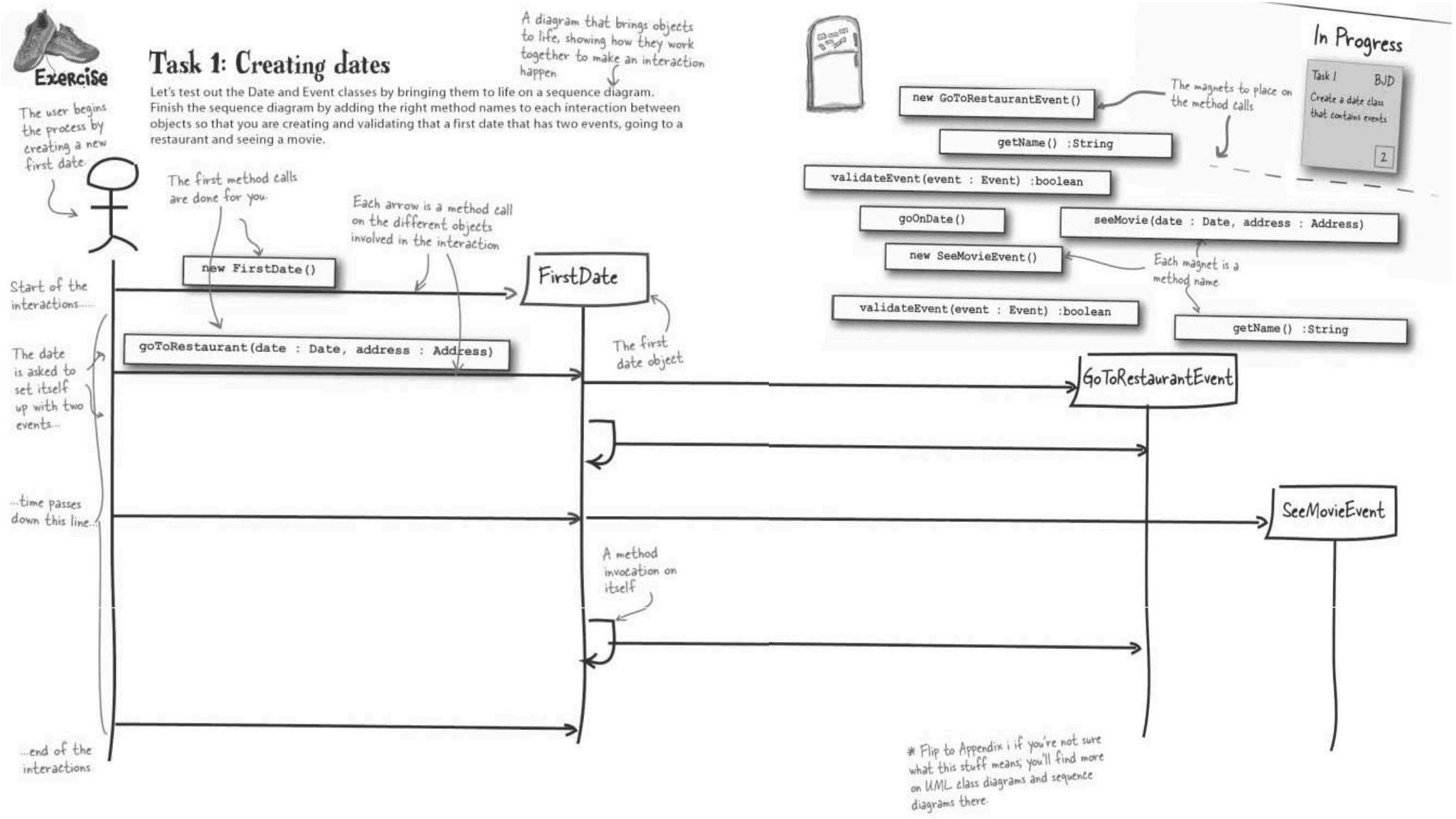
Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram?

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

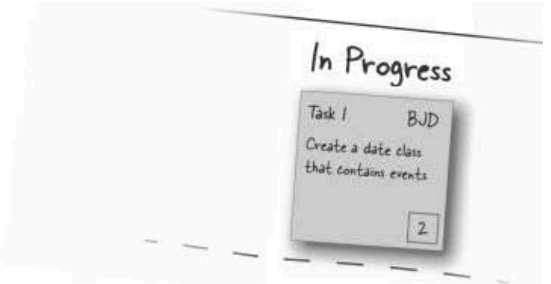
TASK: CREATE THE DATE CLASS

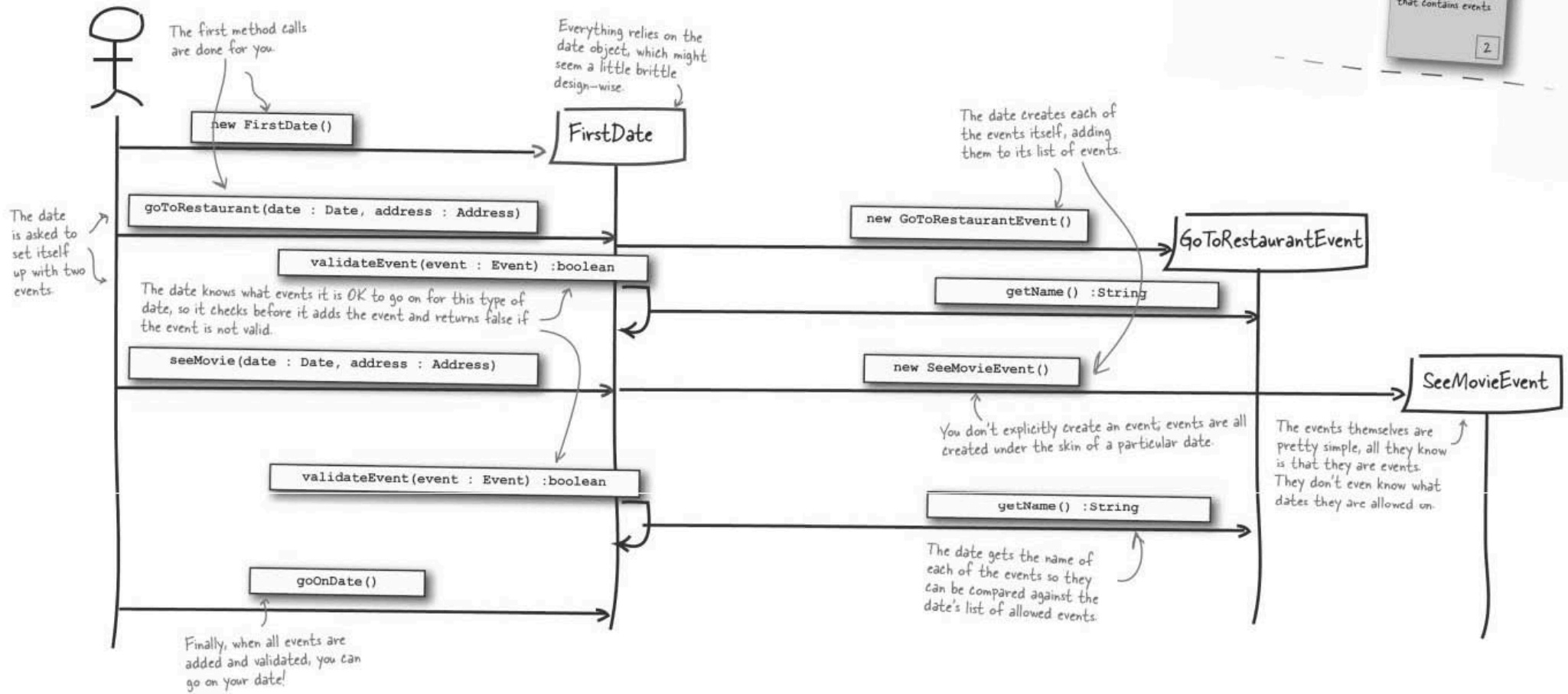
- The Date class is split into three classes, one class for each type of date...
- Each Date can then have a number of Events added to it...



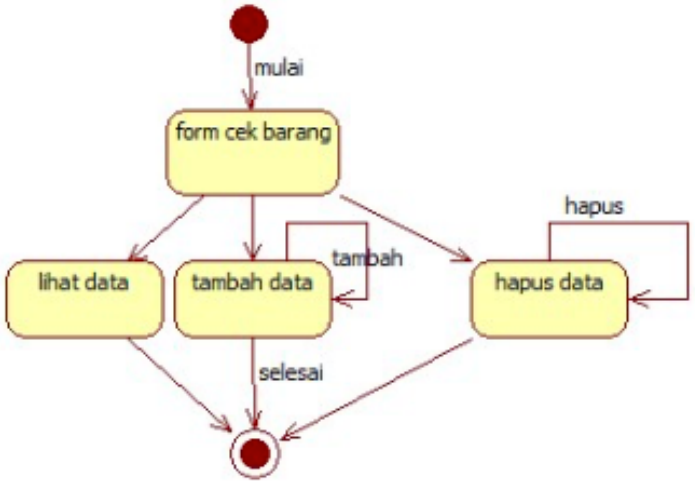

Exercise
Task 1: Creating dates
A diagram that brings objects to life, showing how they work together to make an interaction happen
Let's test out the Date and Event classes by bringing them to life on a sequence diagram. Finish the sequence diagram by adding the right method names to each interaction between objects so that you are creating and validating that a first date that has two events, going to a restaurant and seeing a movie.
The user begins the process by creating a new first date.
The first method calls are done for you.
Each arrow is a method call on the different objects involved in the interaction
Start of the interactions.....
new FirstDate()
FirstDate
The first date object
goToRestaurant(date : Date, address : Address)
The date is asked to set itself up with two events...
GoToRestaurantEvent
...time passes down this line...
SeeMovieEvent
A method invocation on itself
...end of the interactions
In Progress
Task 1 BJD
Create a date class that contains events
2
new GoToRestaurantEvent()
The magnets to place on the method calls
getName() :String
validateEvent(event : Event) :boolean
goOnDate()
seeMovie(date : Date, address : Address)
new SeeMovieEvent()
Each magnet is a method name
validateEvent(event : Event) :boolean
getName() :String
* Flip to Appendix i if you're not sure what this stuff means; you'll find more on UML class diagrams and sequence diagrams there.

Your job was to test out the Date and Event classes by bringing them to life on a sequence diagram. You should have finished the sequence diagram so that you plan and go on a first date with two events, going to a restaurant and seeing a movie.





STATE(?)CHARTS DIAGRAMS



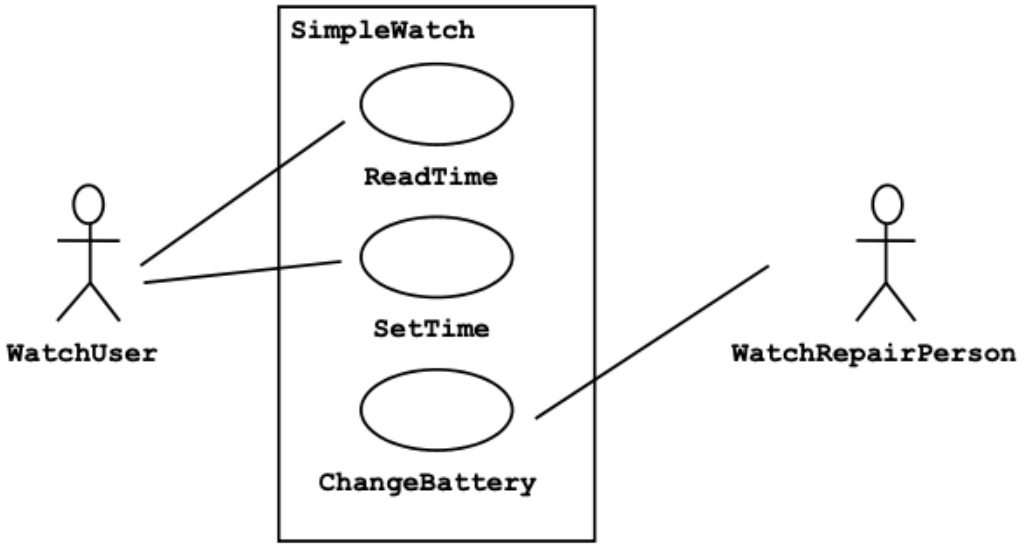
- Represent the behavior of a single object
- States represent different values for the object
- Shows transitions between states as the result of external events, conditions, or time
- Graph whose nodes are states and whose directed arcs are transitions labeled by event names – move from one state to another due to an event

Elements in a State Chart

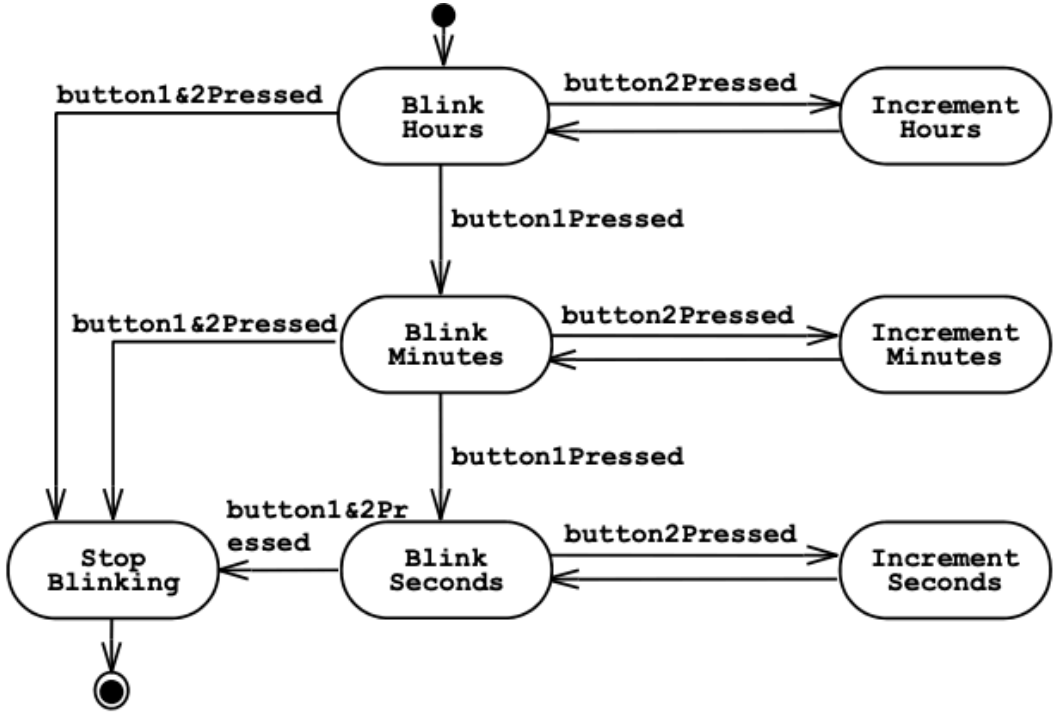
- **States**: Circles containing the expected value of the state variables
- **Transitions**:
  - Arcs from one state to another labeled with the event that causes the transition
  - An event can be a method call or an external action
- **Initial** (start) **state**: is indicated by a solid circle
- **Final** (end) **state**: is indicated by concentric circles – many systems don't have a final state

SIMPLE EXAMPLE

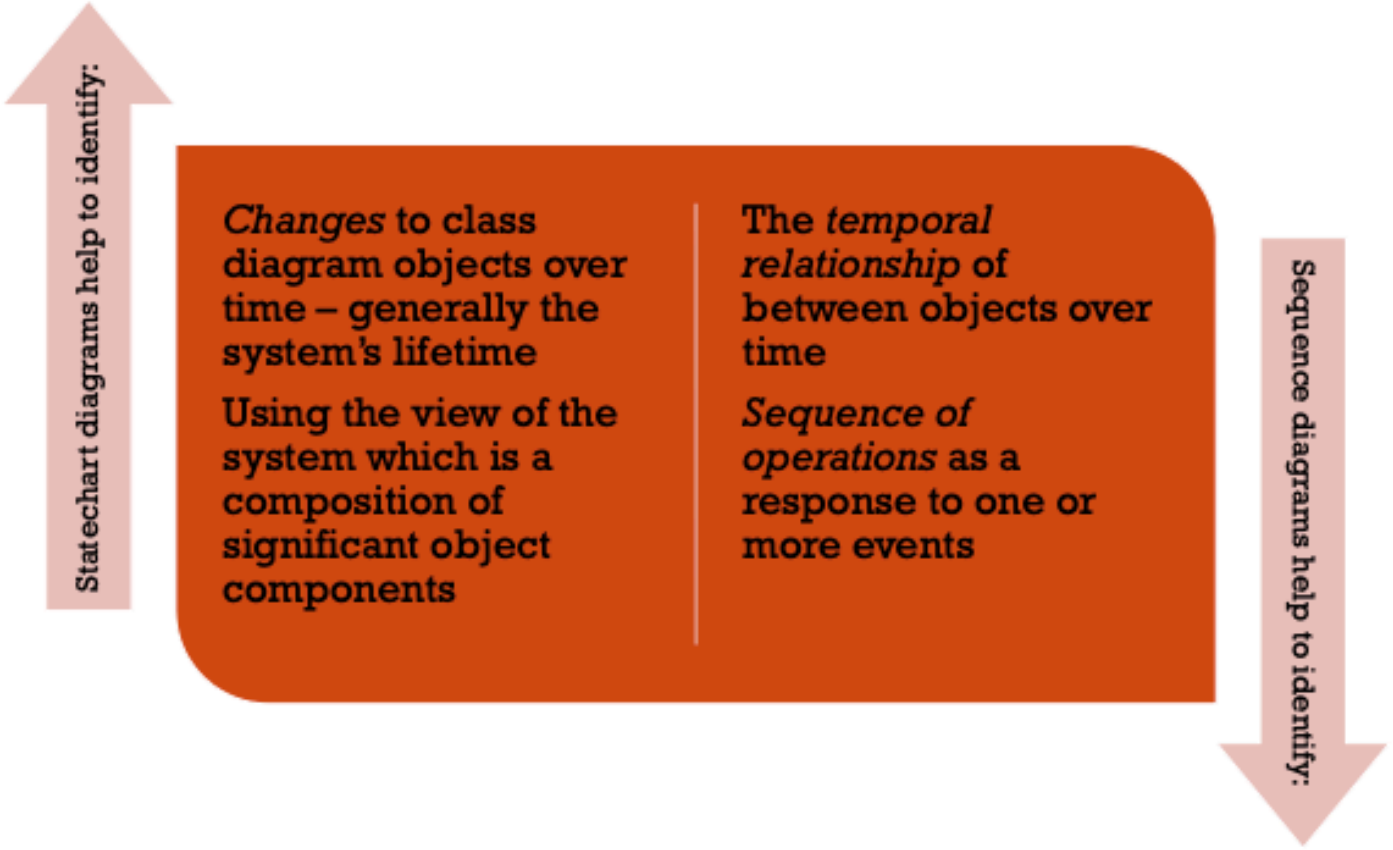
USE CASE MODEL



STATE CHART OF SIMPLEWATCH



STATE CHART DIAGRAM VS. SEQUENCE DIAGRAM



Retrospective Questions

- What is dynamic modeling? Why is it important?
- What is a sequence diagram? What are its elements?
- What is a state diagram? Its elements?
- Difference between state diagram and sequence diagram?
- Which one could you see yourself using? (which seems useful to you)?

MIDTERM INFORMATION

Thursday March 11th, 2021

Virtual (Online) – Two parts

Full class time (and more).

Covers all lectures and textbook readings up to and including Thursday March 4th

- Tuesday March 9th, No class – Wellness Day

Questions from quizzes and assignments could be on it

Short answer, long (diagram) answer

MIDTERM INFORMATION

- Process diagrams - Including phases
- Requirements - Properties, Eliciting, Different Types
- UML diagrams (Use Case, Class, and Sequence)
- UML associations
- Agile Manifesto
- Waterfall v. Incremental
- Use case diagrams
- Different levels of design
- And more!