CS 4350: Fundamentals of Software Engineering
CS 5500: Foundations of Software Engineering

Lesson 2.2 Introduction to UML

Jon Bell, John Boyland, Mitch Wand

Khoury College of Computer Sciences

Learning Objectives for this Lesson

- By the end of this lesson you should be able to:
  - Read and write simple UML class diagrams
  - Illustrate some ways that UML class diagrams may be realized in code
  - Read and write simple UML sequence diagrams

Unified Modeling Language

- UML is a general-purpose visual modeling language developed by an industry consortium in 1997.
- Based on multiple prior visual modeling languages.
- Goal was to have a single standard representation for a large number of SE tasks.
- A large language: 13 different kinds of diagrams
- Currently, UML is at version 2.5.1 (December 2017)

See UML.org and https://www.omg.org/spec/UML/

UML in the context of this course

- We are interested in UML as a human-to-human language.
- So we expect your UML diagrams to "look like" UML diagrams, but we are not interested in every last detail of the notation.
- We just want your diagrams to communicate the important things, with detail as necessary.

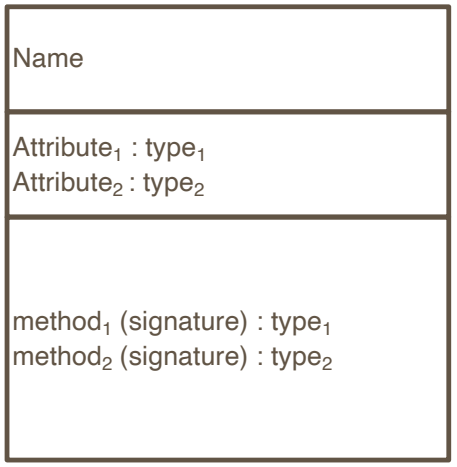
Most common diagram: the Class Diagram

- Class Diagram: Which objects do we need?
  - Which are the features of these objects?
    (attributes, methods)
  - How can these objects be classified?
    (is-kind-of hierarchy, both via inheritance and interface)
  - What associations are there between the classes?

Class Diagrams

- A Class is drawn as a three-part box containing:
  - class name (required)
  - list of attributes with names and types (optional)
  - list of methods with argument lists (optional)

- Components with special roles may be annotated with "stereotypes", which are written with <<...>>.

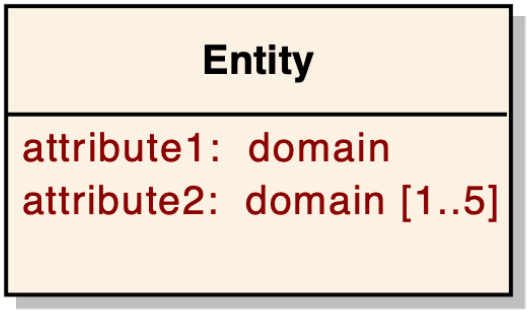


Attributes

- The attributes of a class are roughly those members (or "instance variables" or "properties", depending on what language you are writing in) whose values are either
  - scalars ("simple" attributes)
  - arrays or lists of scalars ("multivalued" attributes)
  - simple structs (e.g. dates or names)
- Class members whose values are full-fledged objects (of this or some other class) are usually represented in UML as **relationships**.

In TypeScript, functions are values, so for us an attribute could have a value that is a function. Your real boss may or may not agree.

Attributes: Example



attribute1 is simple.

attribute2 is multivalued (there can be up to five values stored on attribute2)

domain is UML terminology for "type"

Relationships

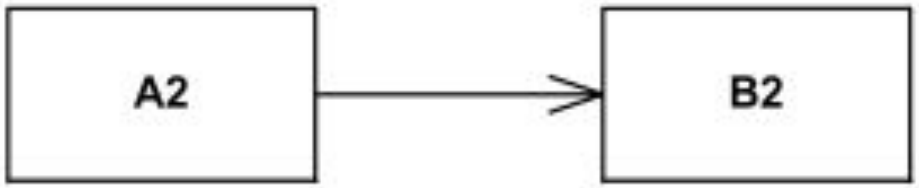
- UML has notations for 3 kinds of relationship between classes:
- Most general relationship: *association*
- Special cases:
  - *Generalization*
  - *Aggregation*

Relationship #1: Association

- An association is a simple semantic relationship between two objects that indicates a link or dependency between them.
- Examples:
  - a portfolio is associated with an investor
  - every sale is associated with the sales representatives that worked on the sale
  - every student is associated with a transcript
- Associations can be directed, meaning there is a relationship from one object to another, or bi-directional, meaning the relationship works both ways.
- Relationships may be annotated with descriptions.
- An association may be implemented in several possible ways.

Properties of Associations: Navigability

- Associations can be navigable, meaning that from one object, you can find the associated object.
- A navigable association is notated with an arrow to indicate the direction in which it flows.
- An association with no arrows means that navigability is unspecified.



from A2, you can get to the associated object of B2

Properties of Associations: Cardinality (or Multiplicity)

- The relationship between two entities has an associated cardinality or multiplicity
  - multiplicity is expressed with specific numbers or ranges,
  - e.g.: 1:1..2 or 1:1..N

- Examples:
  - A student is associated with exactly one transcript (1:1)
    - One student, one transcript.
  - Every course is taught by a professor, but a professor must teach at least one course (1:1..*)
    - One course, one professor. One professor, one or more courses.
  - An address may have a zip code (1:0..1)
    - One address, zero or one zip code

Notation for Cardinality in Associations

Any given instructor teaches 1 course.
Any given course is associated with one instructor.

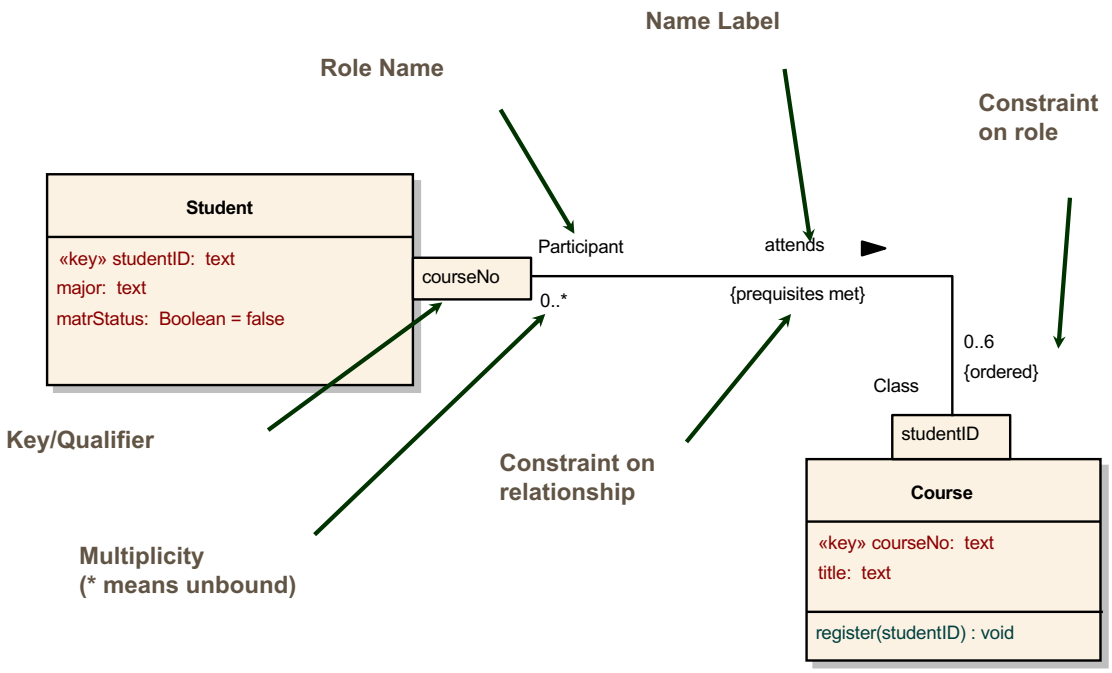
Any given instructor teaches at least 1 and up to 10 courses.
Any given course is associated with one instructor.

Any given instructor teaches 1 or more courses.
Any given course is associated with one instructor.

If no cardina

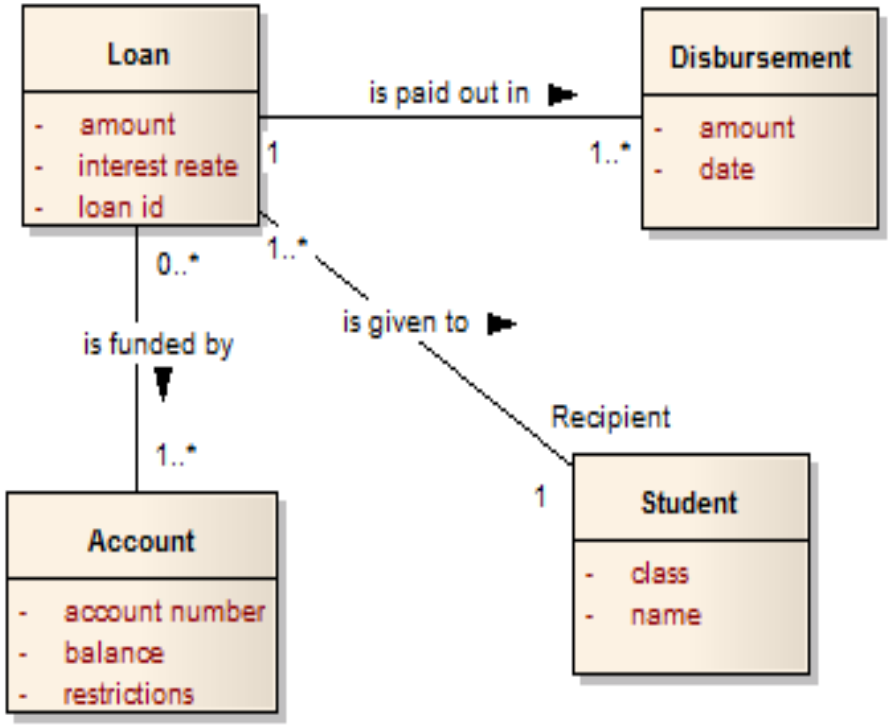
Note: the solid triangle indicates how a human should interpret the relationship ("Instructor teaches Course"). It does not indicate navigability (from an instructor, can you find the list of courses they teach?)

Full Association Specification



The UML folks tried to think of everything you could possibly say about an association. Like much about SE, you only need to memorize the parts you need.

Associations should reflect something about the real world

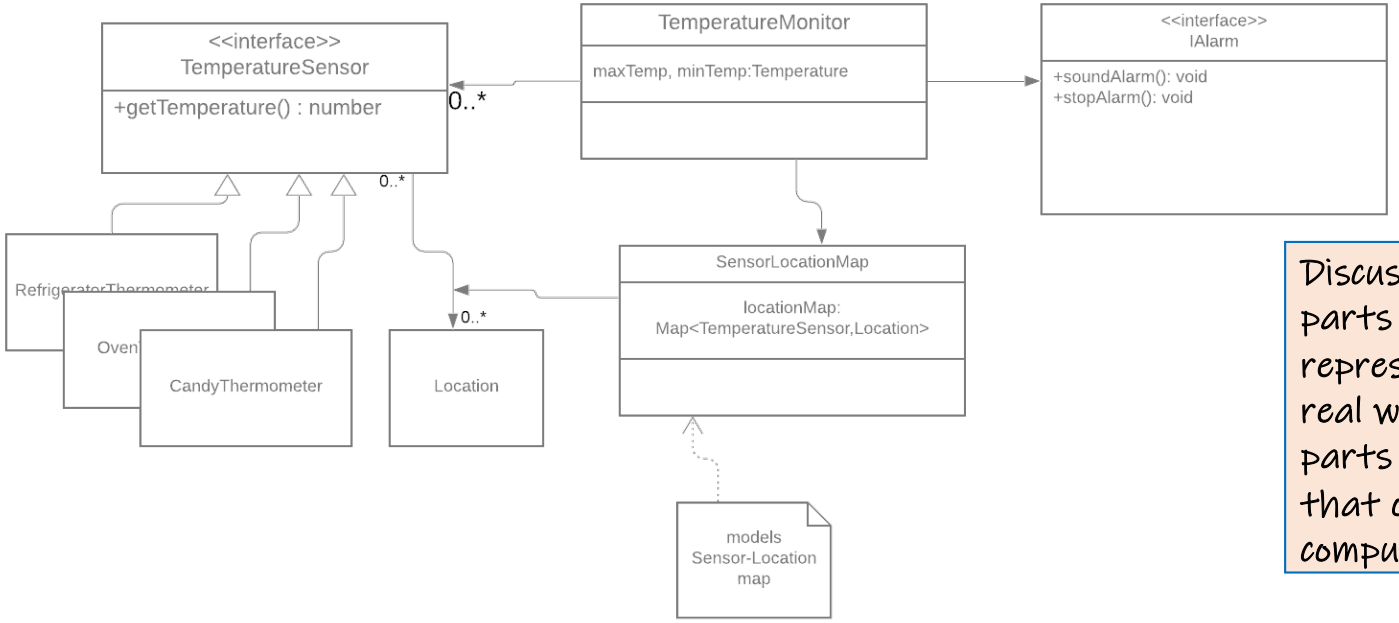


Partial Translation:

We have discovered that a loan can be paid out in multiple disbursements. There does not appear to be any limit to the number of disbursements. In addition, each loan is given to a single student. Apparently, students cannot share loans.

What world are we modeling?

- Sometimes the world we are modeling is not the real world, but the world of entities in our program



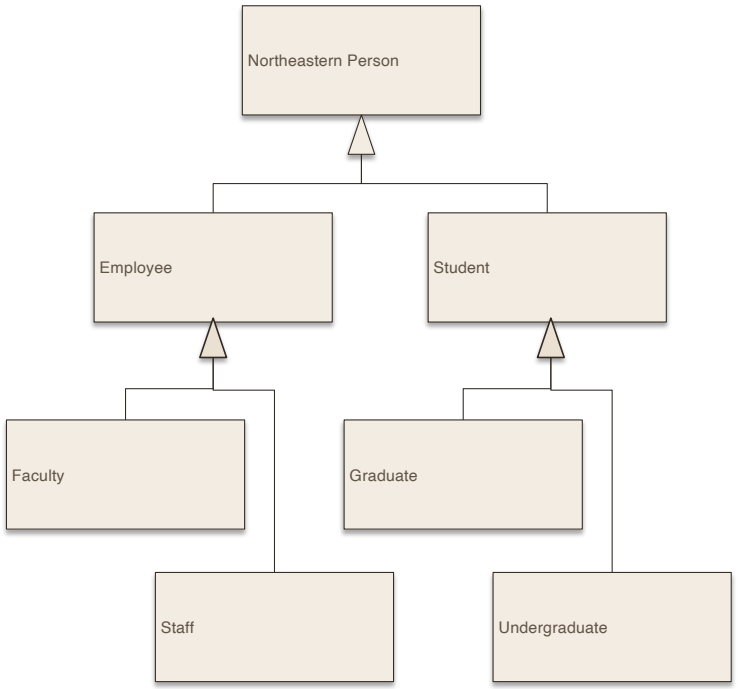
Discussion Question: Which parts of this chart represent things in the real world, and which parts represent things that only live in our computers?

Relationship #2: Generalization

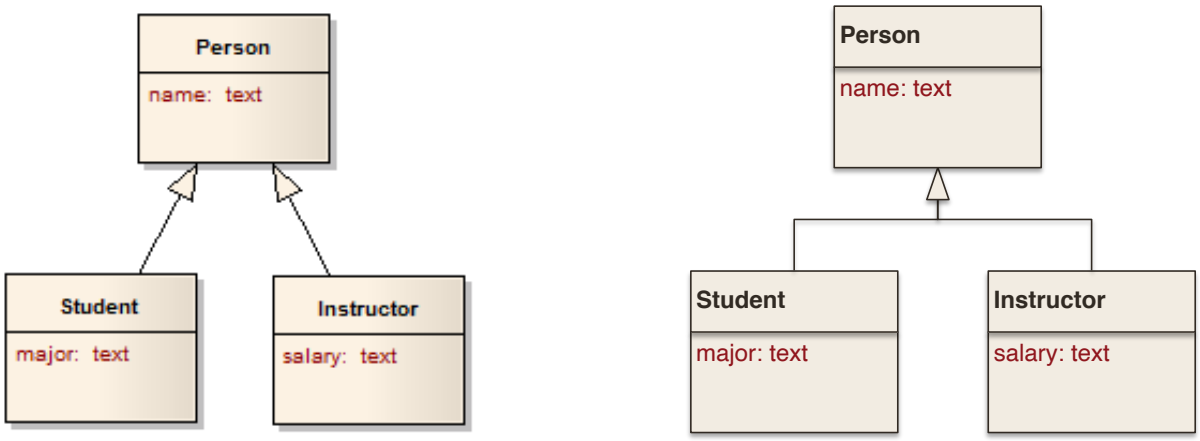
- Generalization is a grouping of entities based on common attributes.
  - describes an is-a-kind-of relationship between entities

Generalization

- more general as you move up
- more specific as you move down
- more specific may inherit attributes and operations from the more general
  - may specialize attributes and operations



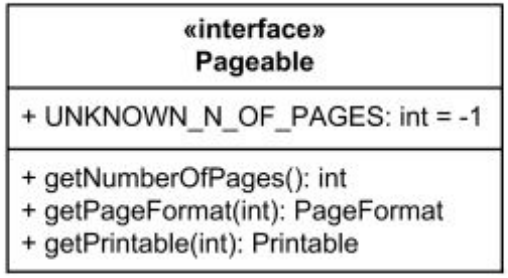
Generalization in UML



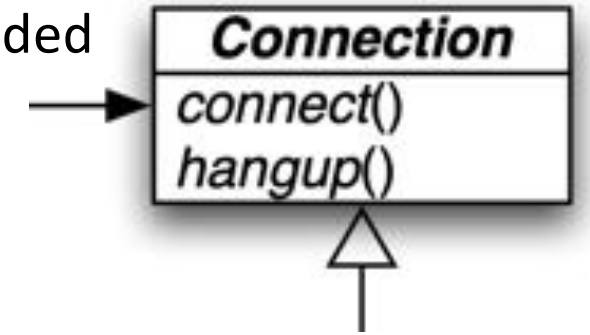
THESE ARE EQUIVALENT

Interfaces and "implements"

- In UML, the "implements" relation is generally considered to be a form of generalization.
- An interface is typically notated like a class, but with the stereotype *<<interface>>*. Alternatively, the name of the interface may be given in italics.
- The "implements" relationship may be notated with a dotted or dashed line, or by an open-headed arrow.

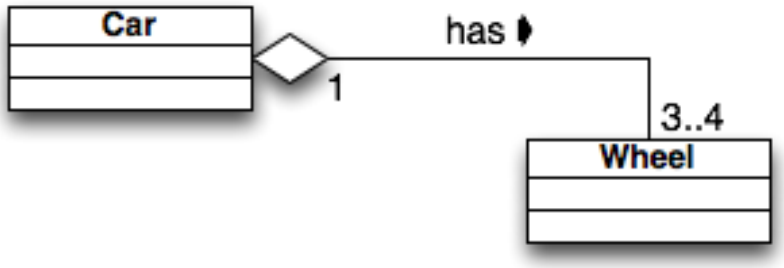


Interface Pageable



Relationship #3: Aggregation

- A car has 3–4 wheels

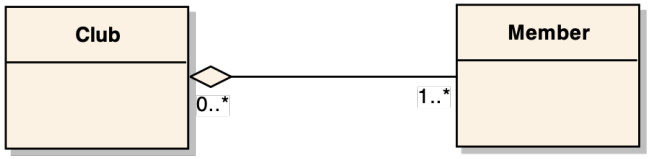


The solid arrow indicates the way we should read "has" (a car "has" wheels, not wheels "has" a car).

Discussion Question: What should the navigability of this association be? Should we be able to get from a Car to the Wheels that it has? Should we be able to get from Wheel to Car?

Aggregation: Definition

- Aggregation is an association that means a “whole/part” or “containment” relationship.
- The distinction between association and aggregation is not always clear.
- Don't stress about this: If in doubt, notate the relationship as a simple association.

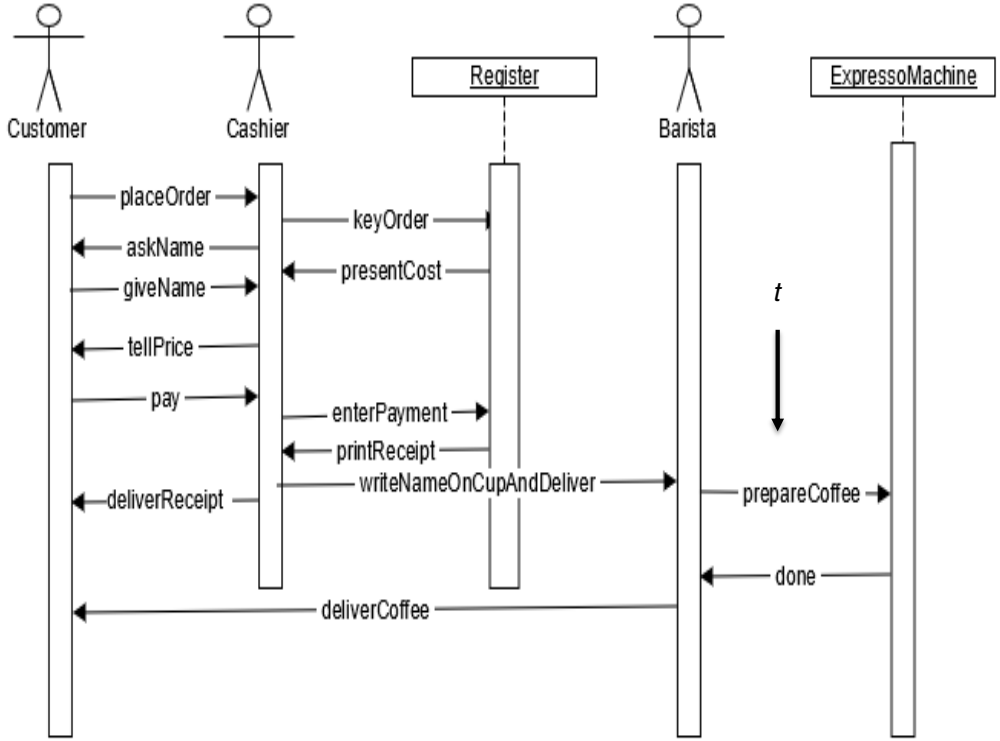


What relation is portrayed in each of these diagrams? What should its navigability be?

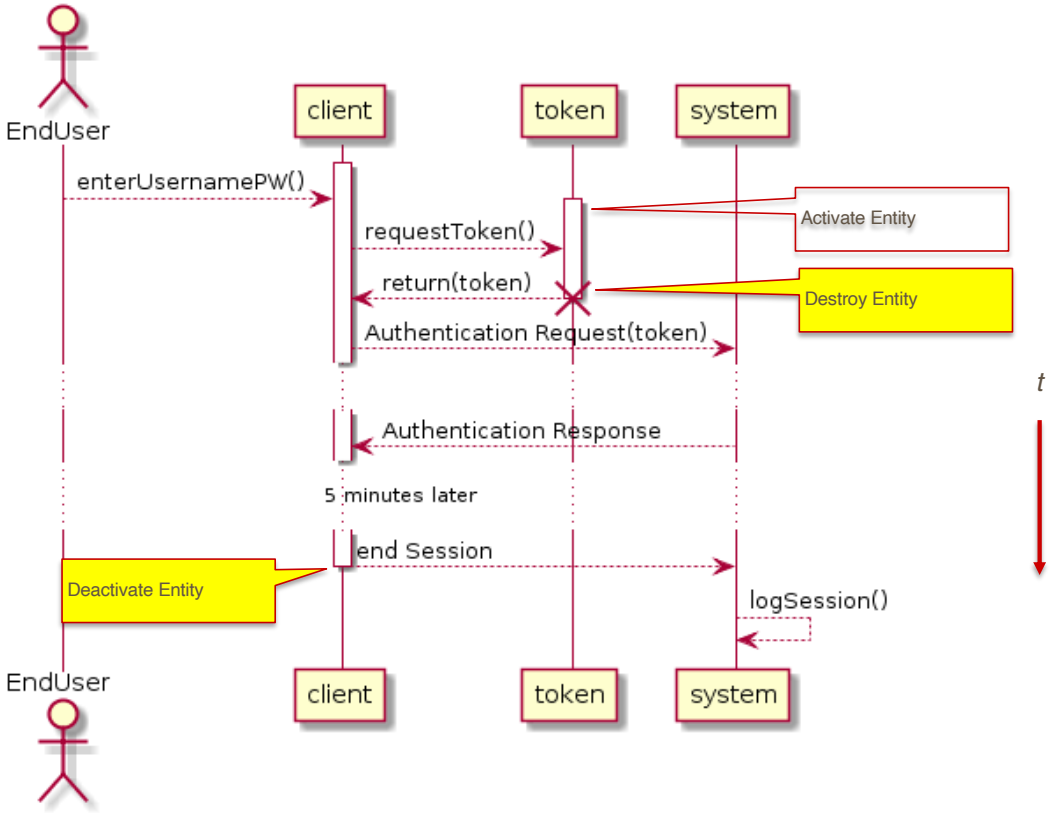
A second kind of UML Diagram: Sequence Diagrams

- Shows the flow between elements of a system (the messaging sequence)
  - Classes (instances of classes)
  - Components
  - Subsystems
  - Actors
- Time is explicitly shown and flows from top to bottom

Example


Customer
Cashier
Register
Barista
ExpressoMachine
placeOrder
keyOrder
askName
presentCost
giveName
t
tellPrice
pay
enterPayment
printReceipt
writeNameOnCupAndDeliver
prepareCoffee
deliverReceipt
done
deliverCoffee

Another Example



Review: Learning Objectives for this Lesson

- At this point you should be able to:
  - Read and write simple UML class diagrams
  - Illustrate some ways that UML class diagrams may be realized in code
  - Read and write simple UML sequence diagrams

Next steps...

- Come to class prepared with questions!
- In our next lessons, we will explore design patterns, which are yet another language for explaining objects and their interactions.