CS 4530
Software Engineering

Lecture 2 - Design Documentation: CRC + UML

Jonathan Bell, John Boyland, Mitch Wand
Khoury College of Computer Sciences

Zoom Mechanics

- Recording: This meeting is being recorded
- If you feel comfortable having your camera on, please do so! If not: a photo?
- I can see the zoom chat while lecturing, slack while you're in breakout rooms
- If you have a question or comment, please either:
  - "Raise hand" - I will call on you
  - Write "Q: <my question>" in chat - I will answer your question, and might mention your name and ask you a follow-up to make sure your question is addressed
  - Write "SQ: <my question>" in chat - I will answer your question, and not mention your name or expect you to respond verbally

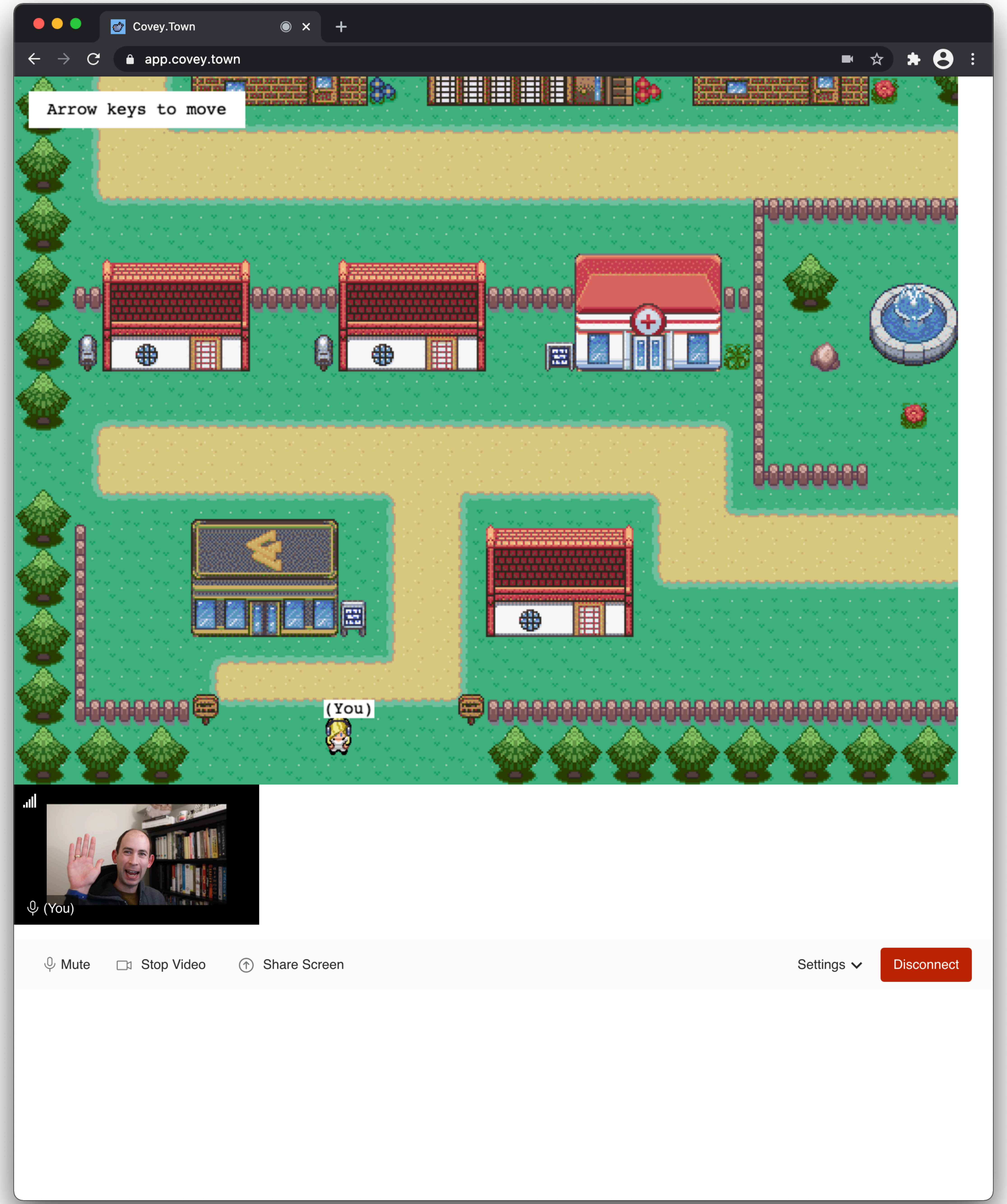
Today’s Agenda

HW1 Discussion

Documenting designs with CRC + UML diagrams

Activity: UML

Discussion: HW1



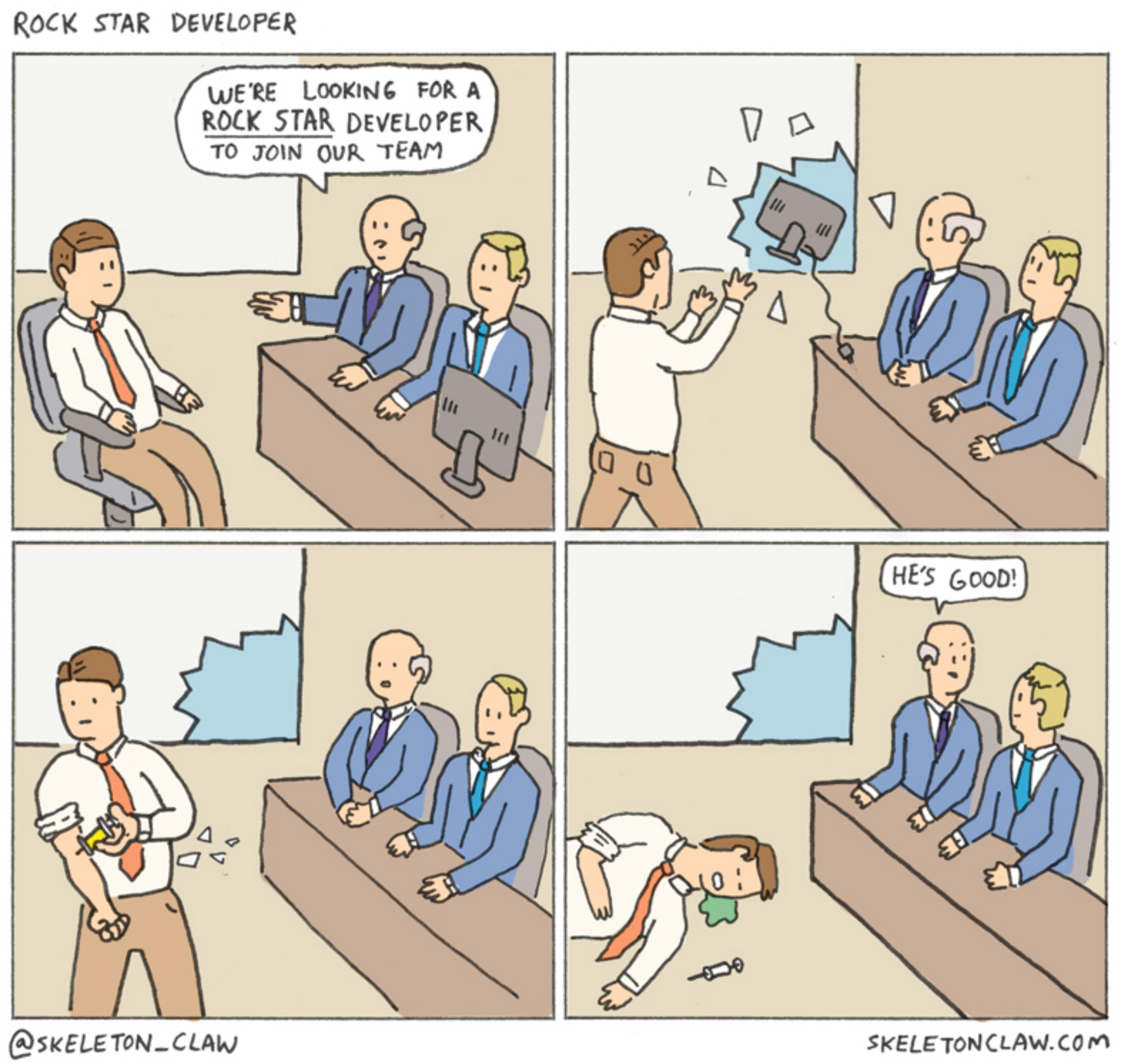
“The 10x Engineer”

AKA “The Rock-Star Engineer,” “The Ninja Developer”

What makes a 10x Developer?

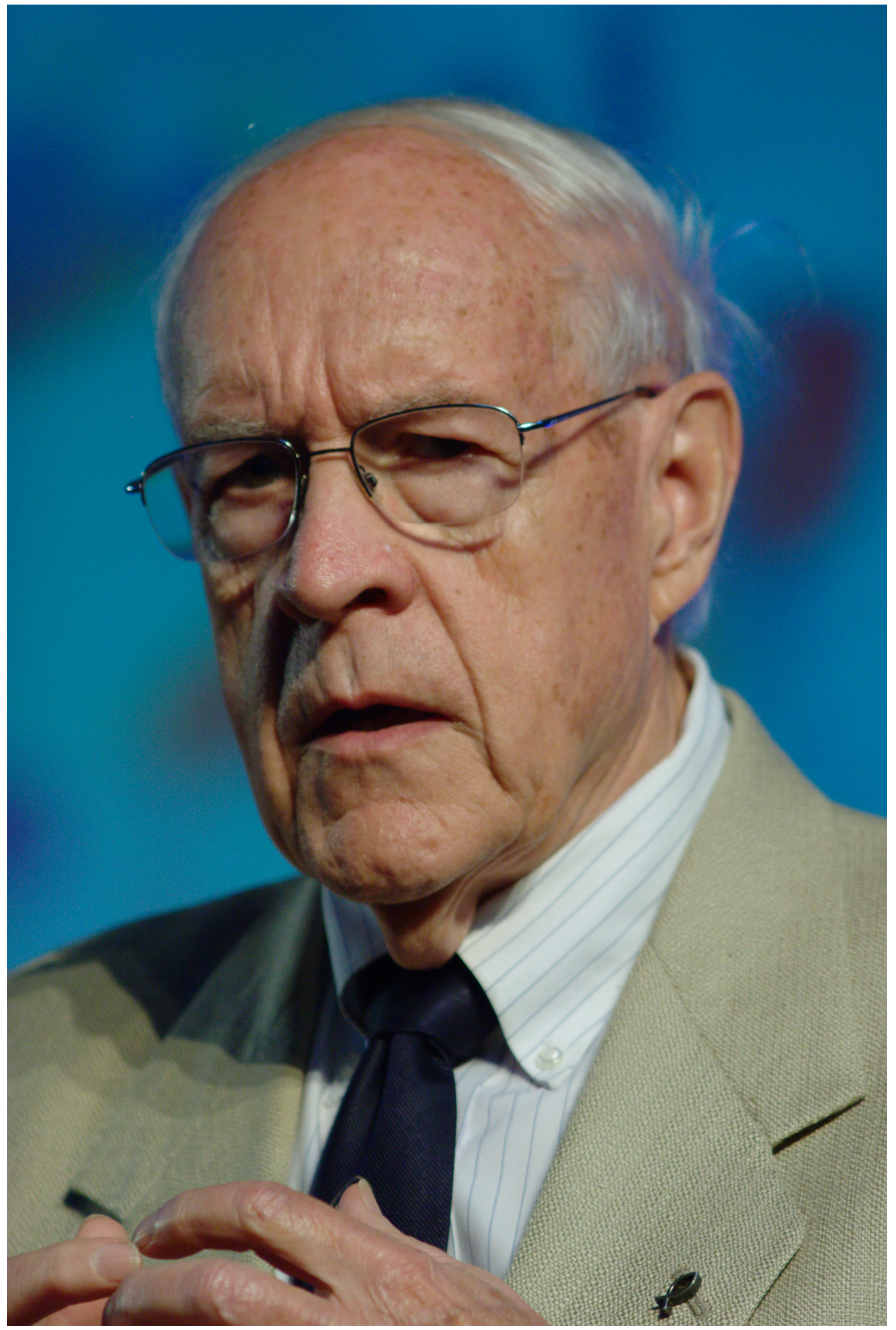
#10xdeveloper #productivity #beginners #career

Davide de Paolis Mar 11, 2019 · 6 min read



Conceptual Design is Hard

There is “No Silver Bullet” for a 10x improvement



“The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions. … I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.”

Fred Brooks, 1987

Collaboration: The 10x Team

A “Dream Team,” if you will…

- “Many eyes make all bugs shallow” (ancient proverb)
- Avoid a single point of failure

The 10x Team, or the 1/10 Team?

Mythical Man-Month: “adding manpower to a late software project makes it later”

- Knowledge sharing needs to scale linearly (or sub linearly) with org growth:
  - Mentorship
  - Q&A
  - Mailing lists
  - Tech talks
  - Documentation <— Our focus today

For more on knowledge sharing in teams, see SE@Google Ch 3

Design Documents

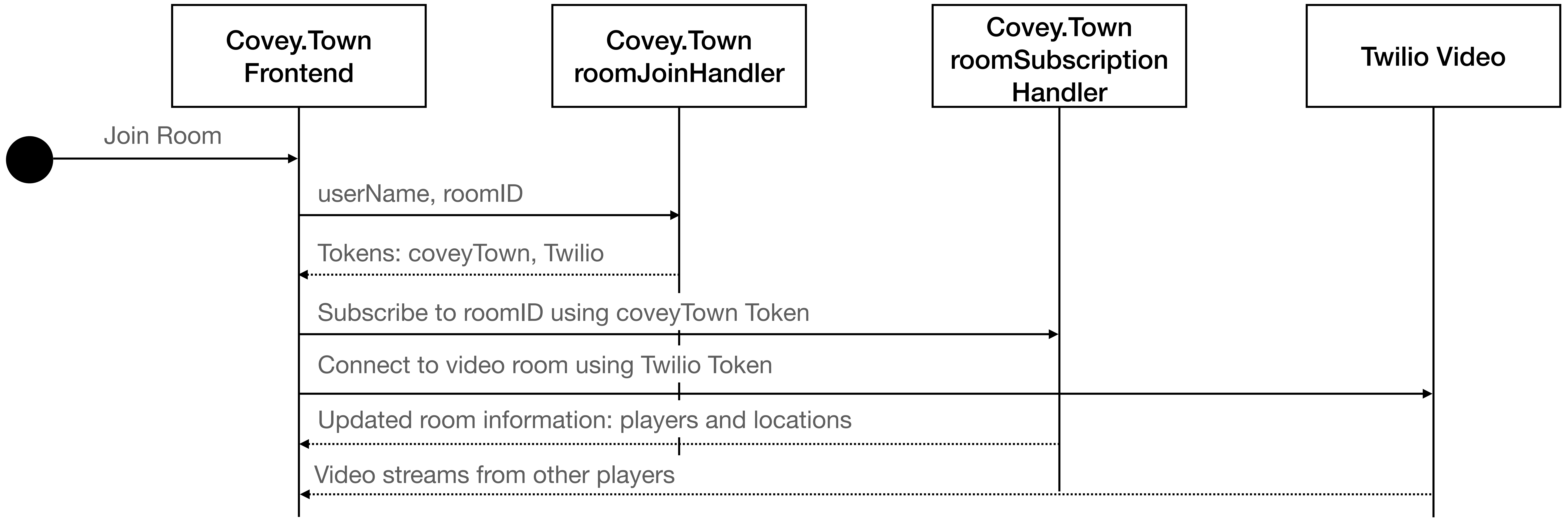
Why?

- At design time:
    - Consider alternative solutions
    - Identify flaws
- At implementation/debugging time:
    - A handy reference
- Design documents include…
    - Goals of design
    - Implementation strategy
    - Discussion of alternative designs and their strong and weak points

Design Diagrams - UML Sequence

Improving understanding and understandability

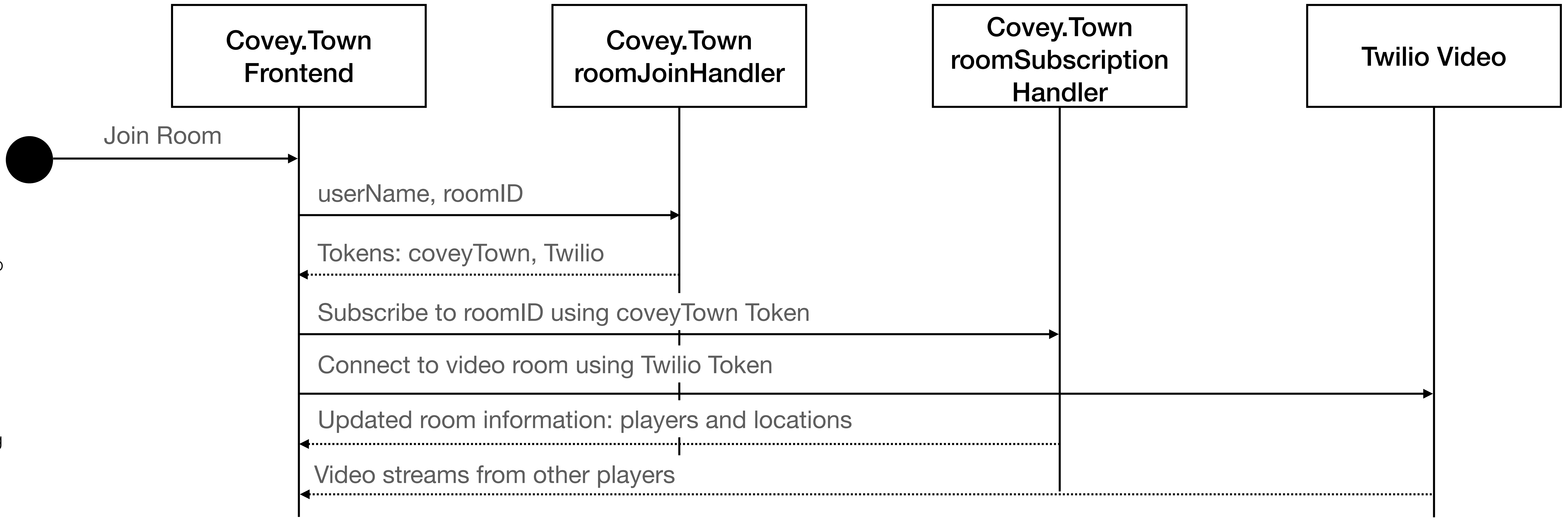
```
/**
 * A handler to process a player's request to join a room. The flow is:
 *  1. Client makes a RoomJoinRequest, this handler is executed
 *  2. Client uses the sessionToken returned by this handler to make a subscription to the room,
 *  @see roomSubscriptionHandler for the code that handles that request.
 *
 * @param requestData an object representing the player's request
 */
export async function roomJoinHandler(requestData: RoomJoinRequest): Promise<RoomJoinResponse>
```



Covey.Town Frontend
Covey.Town roomJoinHandler
Covey.Town roomSubscription Handler
Twilio Video
Join Room
userName, roomID
Tokens: coveyTown, Twilio
Subscribe to roomID using coveyTown Token
Connect to video room using Twilio Token
Updated room information: players and locations
Video streams from other players

Design Diagrams - UML Sequence

Improving understanding and understandability

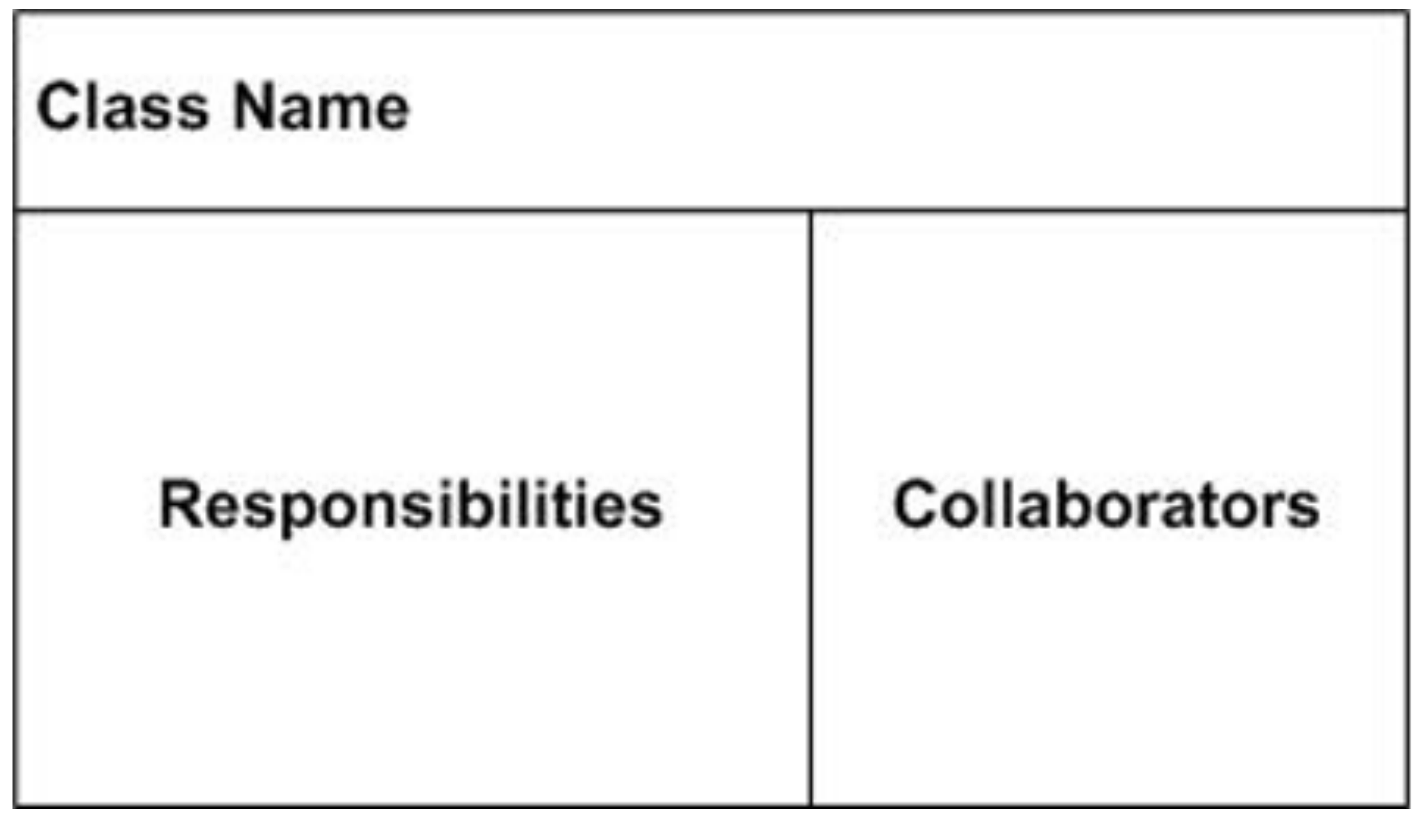

Covey.Town Frontend
Covey.Town roomJoinHandler
Covey.Town roomSubscription Handler
Twilio Video
Join Room
userName, roomID
Tokens: coveyTown, Twilio
Subscribe to roomID using coveyTown Token
Connect to video room using Twilio Token
Updated room information: players and locations
Video streams from other players

Draw a line for each message, with an arrowhead pointing to the receiving class instance

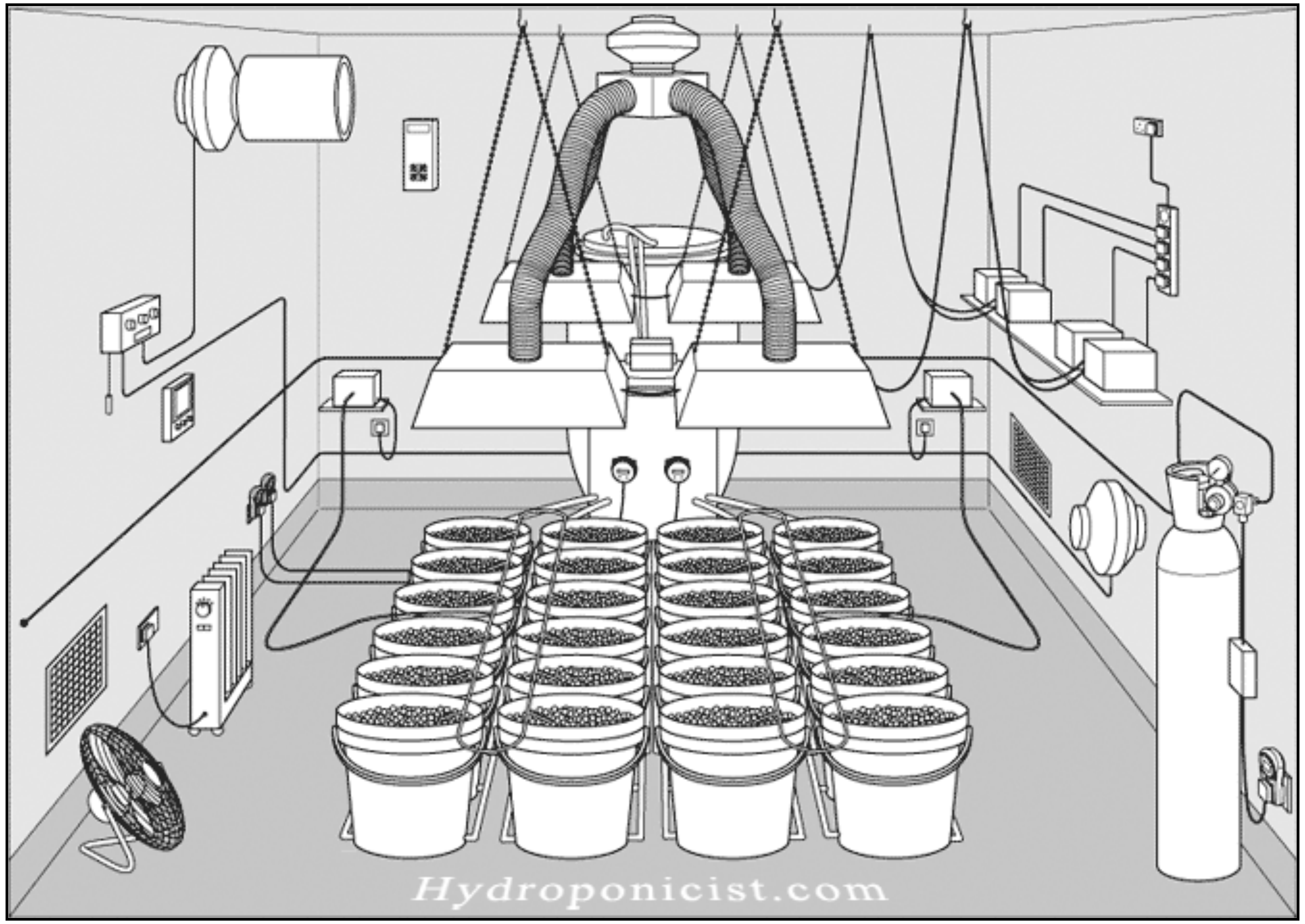
Draw a dotted line for each return value, with an arrowhead pointing back to the originating class instance

CRC Cards

- Each class is a thing, entity or object
- Each responsibility is some action the entity needs to do
- Collaborators are other classes the entity interacts with, communicates with, contains, knows about, or that otherwise help it perform one or more responsibilities
- CRC focuses on the purpose of each entity rather than its processes, data flows and data stores (procedural design)



Example: Sensors



CRC Card for TemperatureSensor

```
// temperatures are measured in Celsius
type Temperature = number

interface TemperatureSensor {
    // return the current temperature
    // at the sensor location
    getTemperature () : Temperature
}
```

CRC cards are supposed to be informal, so don't get hung up on emulating the exact words or the exact layout I've used here.

Class Name: TemperatureSensor (interface)	
State: none	
Responsibilities	Collaborators
establish interface for thermometers in the system	RefrigeratorThermometer
	OvenThermometer
	etc.
	TemperatureMonitor

TemperatureMonitor (1)

```
class TemperatureMonitor {
    constructor(

        // the sensors
        private sensors: TemperatureSensor[],

        // map from sensor to its location
        private sensorLocationMap: SensorLocationMap,

        private maxTemp: Temperature,
        private minTemp: Temperature,
        private alarm: IAlarm,
    ) { }

    // sensor in range?
    private isSensorInRange (sensor:TemperatureSensor) : boolean {
        const temp: Temperature = sensor.getTemperature()
        return ((temp < this.minTemp) || (temp > this.maxTemp))
    }
```

Here's a slightly more elaborate TemperatureMonitor

It monitors multiple sensors

And it knows where each sensor is

Better division into one method/one job than our earlier version.

TemperatureMonitor (2)

```
    // if the any of the sensors is out of range, sound the alarm
    public checkSensors(sensor:TemperatureSensor): void {
        this.sensors.forEach(sensor => {
            if (!(this.isSensorInRange(sensor))) {
                this.soundAlarm(sensor)
            }
        })
    }

    private soundAlarm (sensor) {
        const location = this.sensorLocationMap.getLocation(sensor)
        this.alarm.soundAlarm(location)
        }

}
```

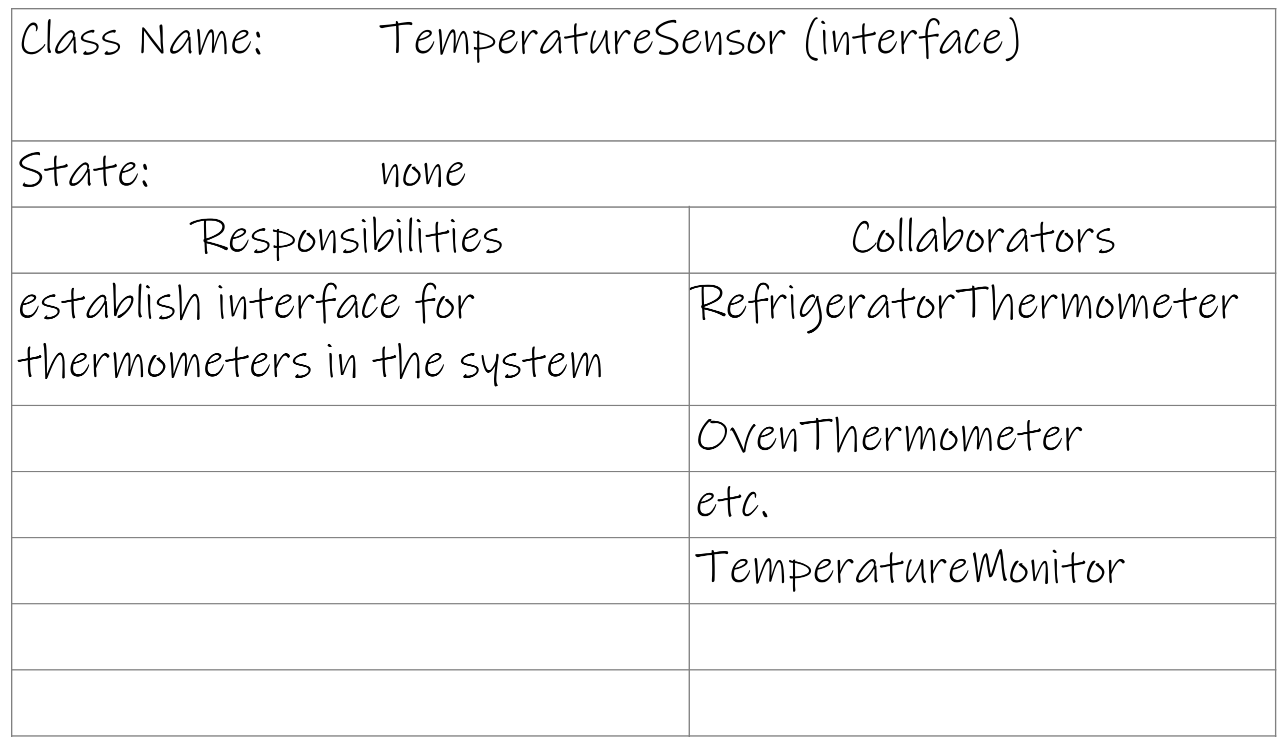
CRC Card for TemperatureMonitor

Class Name:	TemperatureMonitor
State:	sensors, maxTemp, minTemp, alarm
Responsibilities	Collaborators
if any of the sensors is out of range, tell the alarm to sound at its location	TemperatureSensor
	SensorLocationMap
	IAlarm

CRC Cards: Where to start?

Building the cards

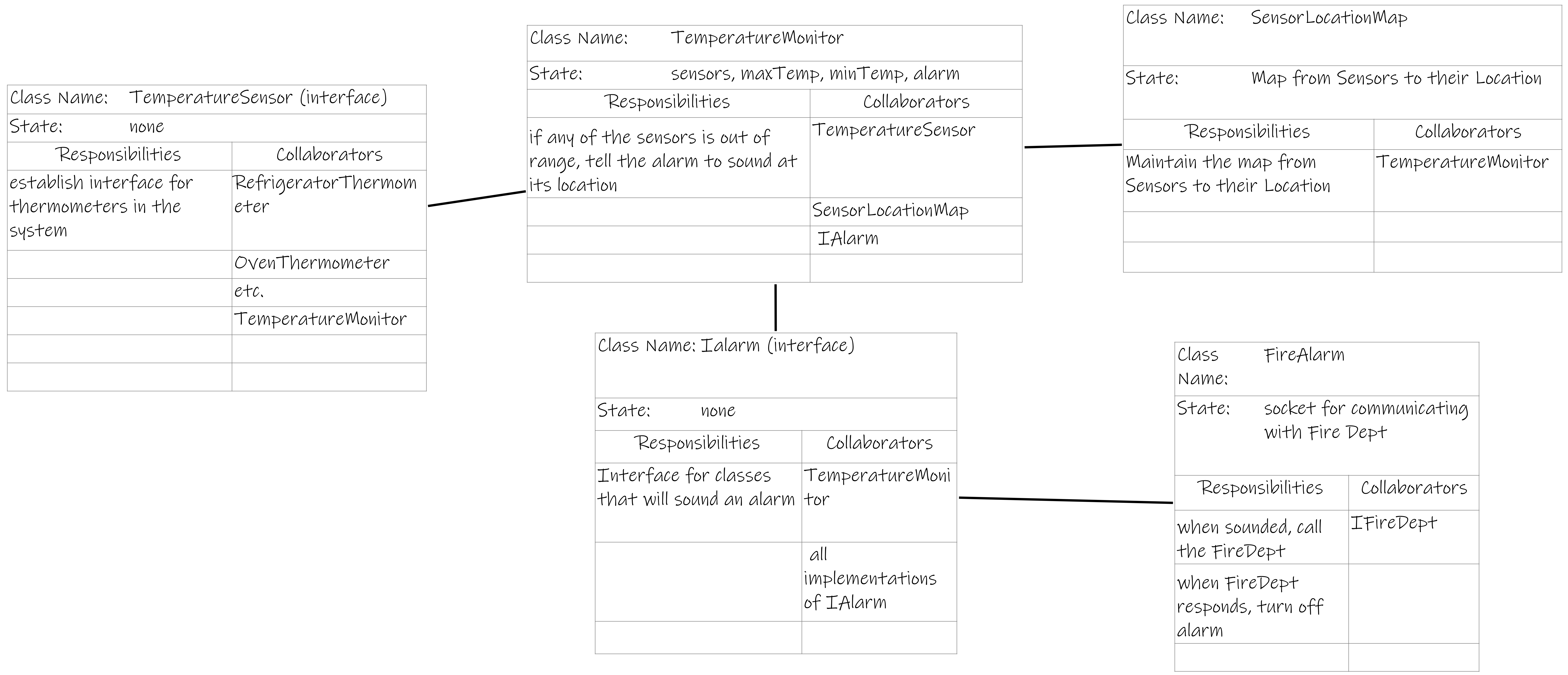
- Find the nouns: entities that “do” actions (classes)
- Find the verbs: what gets done, not how (responsibilities)
- Find the relationships



Class Name:	TemperatureSensor (interface)
State:	none
Responsibilities	Collaborators
establish interface for thermometers in the system	RefrigeratorThermometer
	OvenThermometer
	etc.
	TemperatureMonitor

CRC Cards: Putting them to use

Not just static objects!



Class Name:	TemperatureSensor (interface)
State:	none
Responsibilities	Collaborators
establish interface for thermometers in the system	RefrigeratorThermometer
	OvenThermometer
	etc.
	TemperatureMonitor

Class Name:	TemperatureMonitor
State:	sensors, maxTemp, minTemp, alarm
Responsibilities	Collaborators
if any of the sensors is out of range, tell the alarm to sound at its location	TemperatureSensor
	SensorLocationMap
	IAlarm

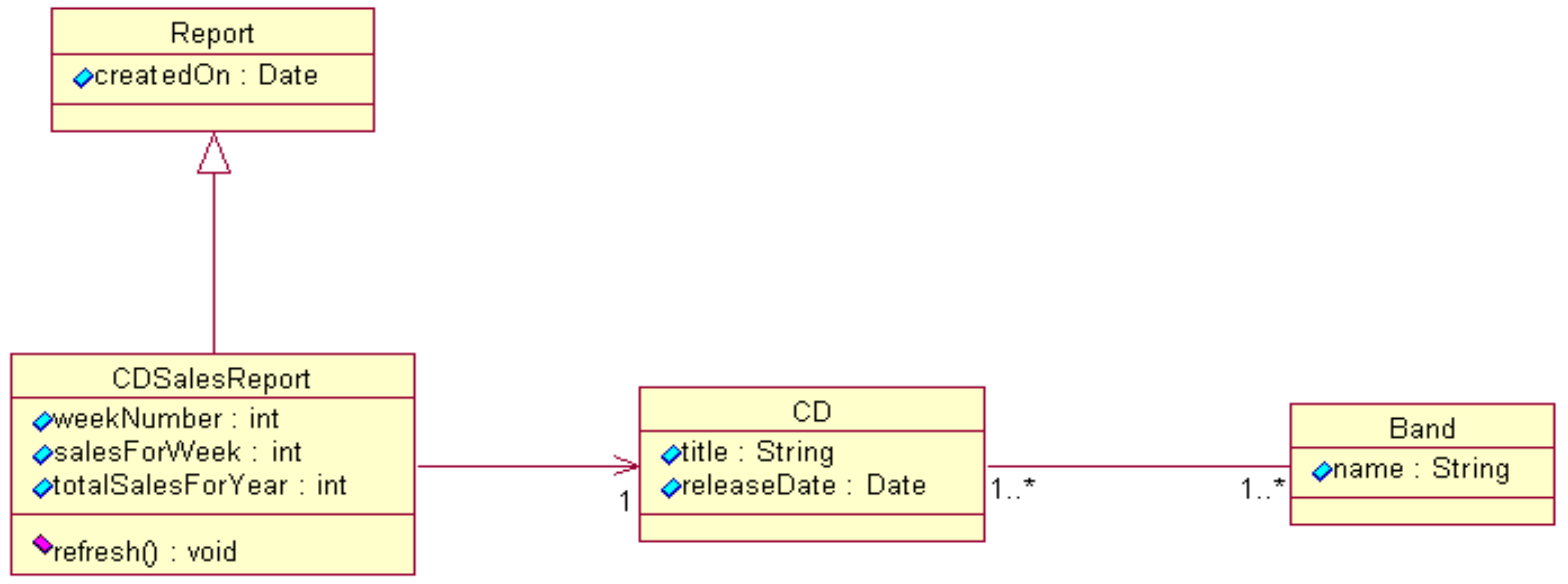
Class Name:	SensorLocationMap
State:	Map from Sensors to their Location
Responsibilities	Collaborators
Maintain the map from Sensors to their Location	TemperatureMonitor

Class Name:	Ialarm (interface)
State:	none
Responsibilities	Collaborators
Interface for classes that will sound an alarm	TemperatureMonitor
	all implementations of IAlarm

Class Name:	FireAlarm
State:	socket for communicating with Fire Dept
Responsibilities	Collaborators
when sounded, call the FireDept	IFireDept
when FireDept responds, turn off alarm	

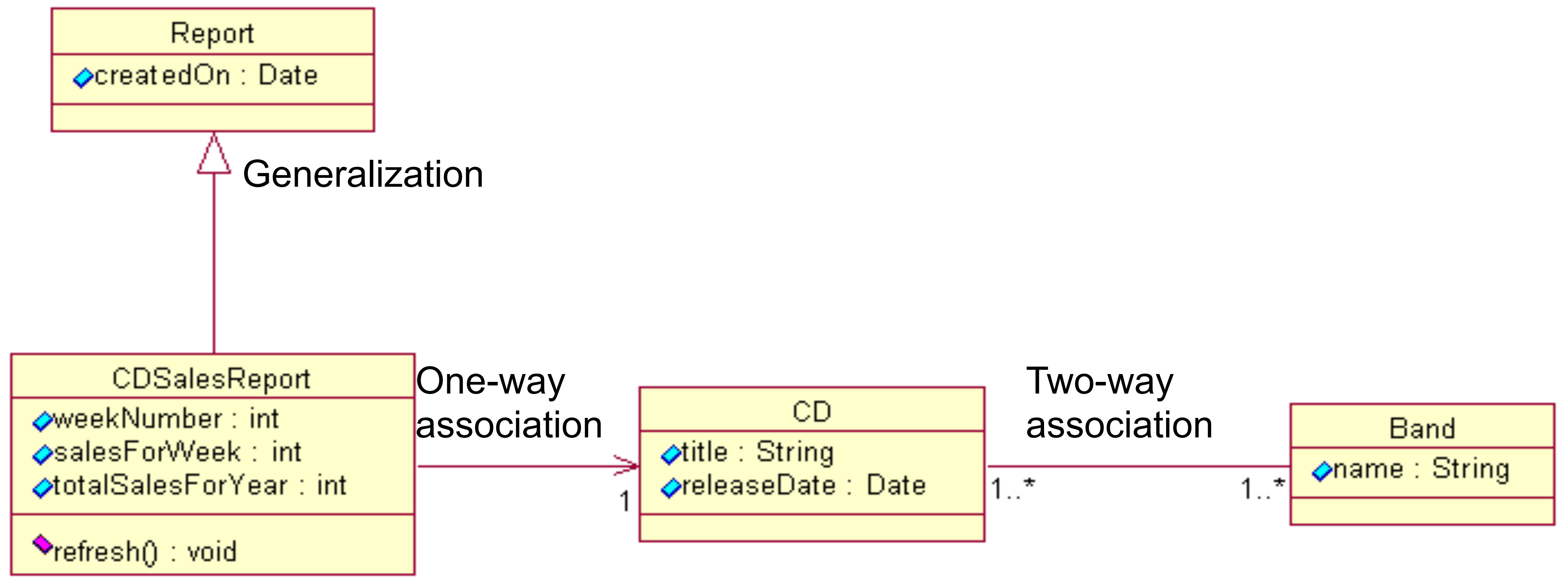
UML Class Diagrams

- Graphically shows relationships between entities
- Not necessarily a 1:1 correspondence to code: good for domain modeling
- Example: reporting on compact disc sales

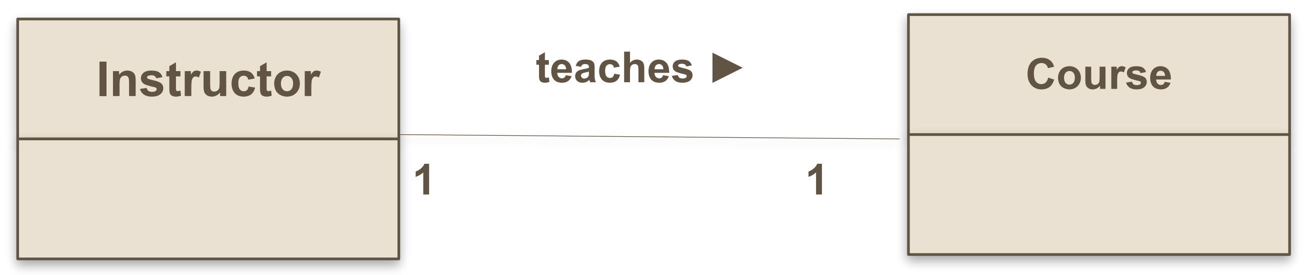


UML Class Diagrams

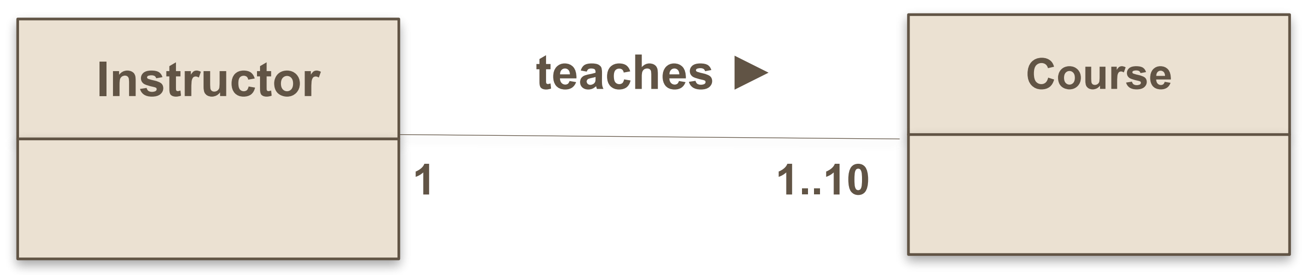
- Indicate relationships using different kind of arrows:
  - Generalization (is a)
  - Association (has a)



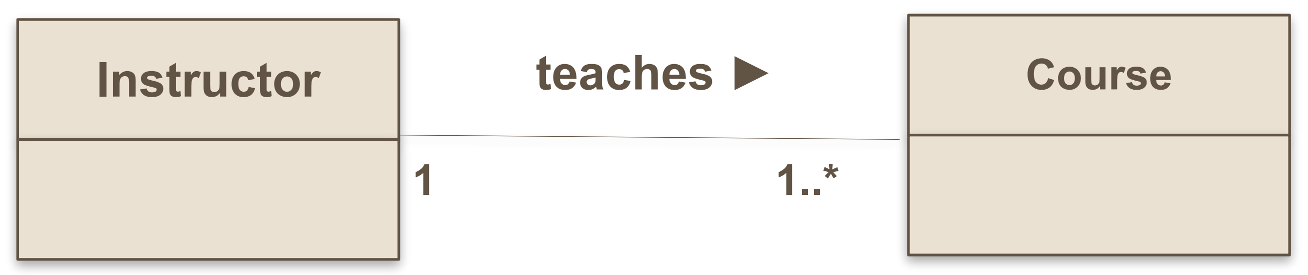
UML Class Diagrams: Cardinality



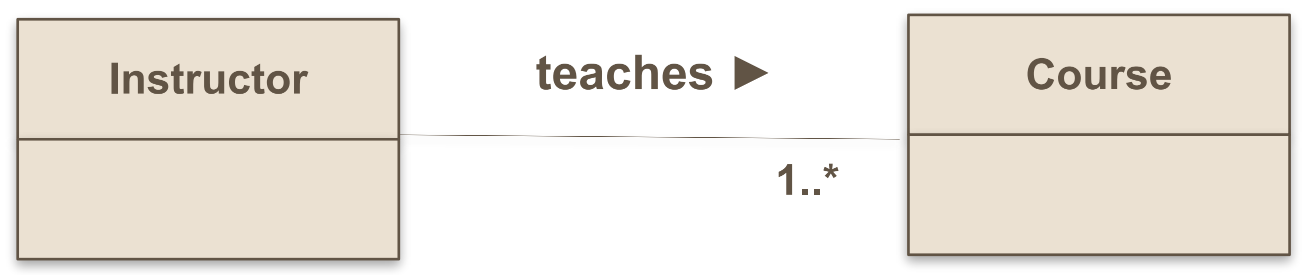
Any given instructor teaches 1 course.
Any given course is associated with one instructor.



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



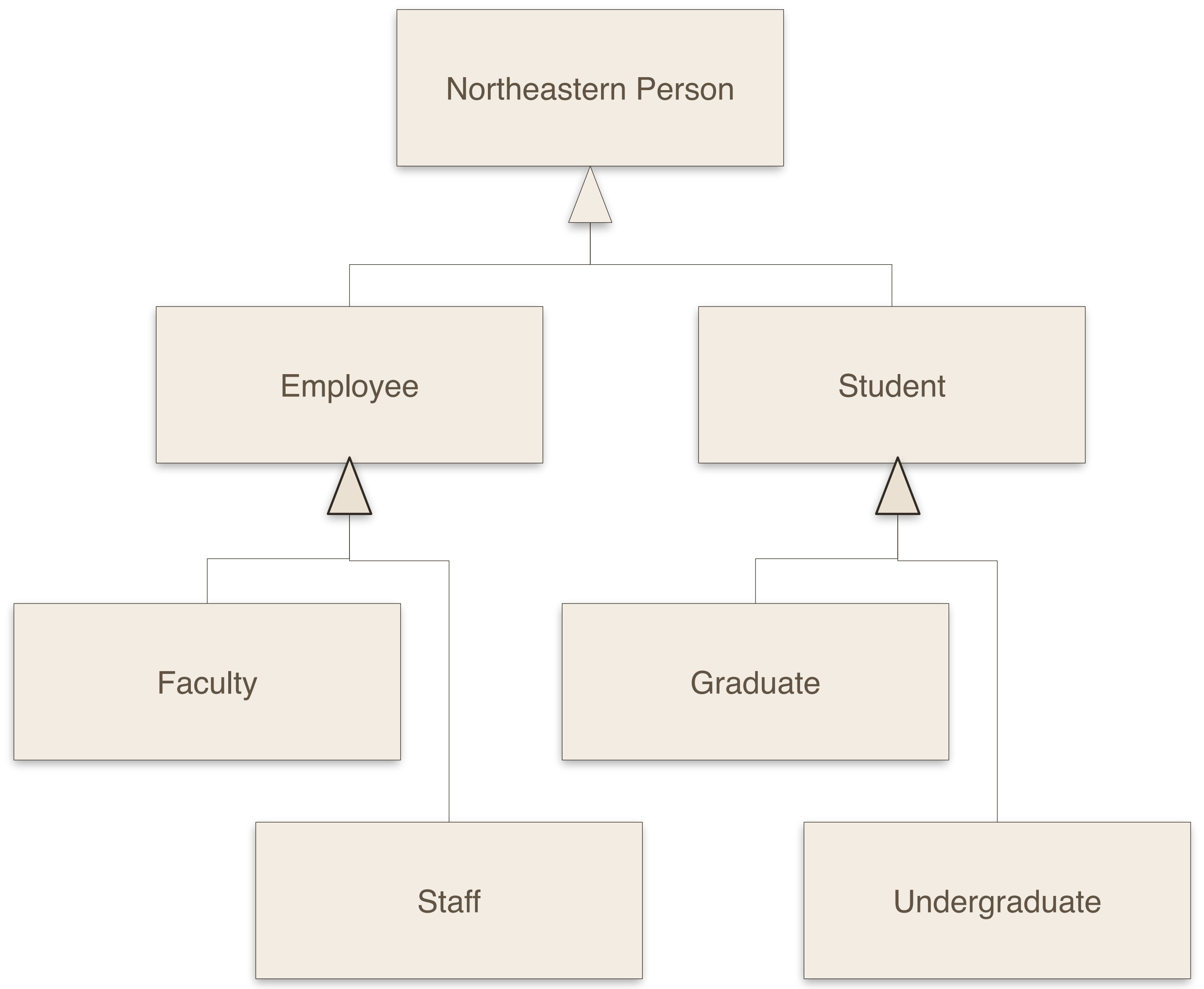
Any given instructor teaches 1 or more courses.
Any given course is associated with one instructor.



If no cardinality is specified, it defaults to 1.

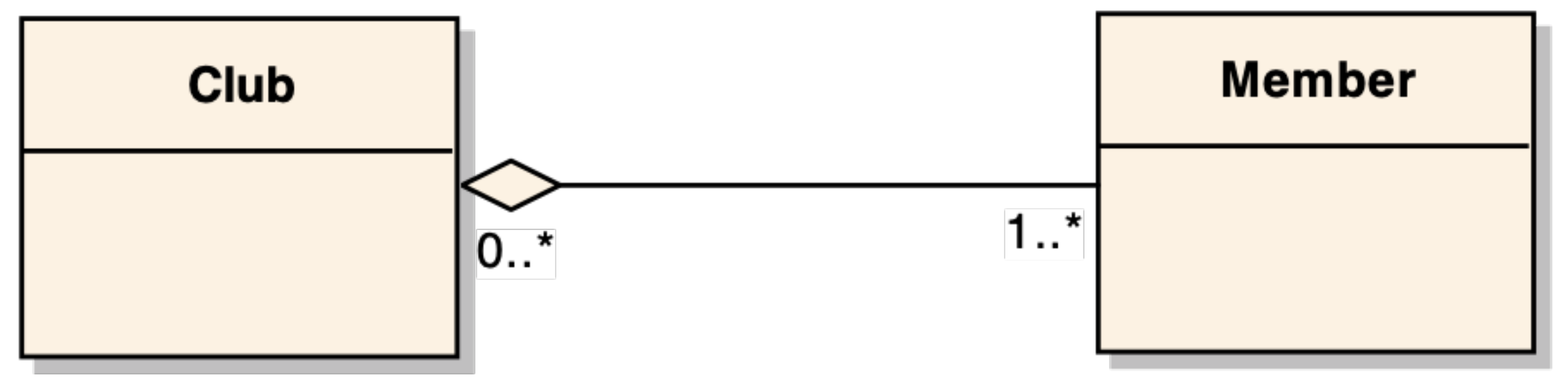
UML Class Diagrams: Generalization

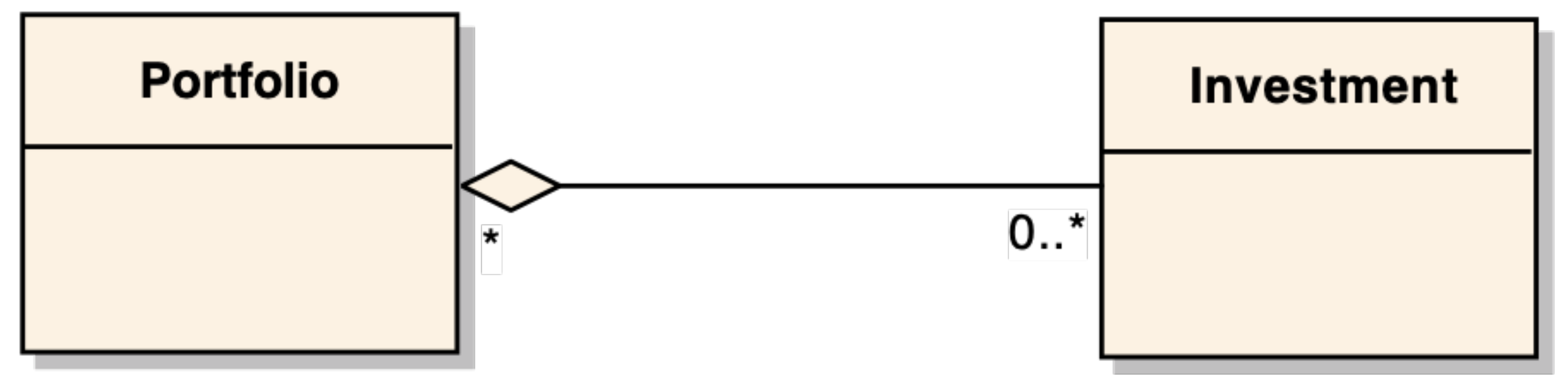
- more generic as you move up
- more specific as you move down
- more specific inherits attributes
  and operations from the more general
  - may specialize attributes and operations



UML Class Diagrams: Aggregation

- 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.





UML Activity: TVM

Ticket Vending Machines

- TVMs accept cash and credit cards as payments to sell fares, which are loaded onto passes
- TVMs sell two kinds of fares:
  - Time-based fares
  - Value-based fares
- Fares can be loaded onto passes, passes can be:
  - CharlieCard
  - CharlieTicket
- Your task: Create a UML class diagram that represents:
  - The TVM itself; the two kinds of fares; the two kinds of passes