
Full Stack Development
Containers, Microservices and UI
4. Code Design

What is Code Design

- This module is not about how to write code
  - This is not a programming course
- This module is about **deciding what code to write**
  - The classes and packages we chose to use
  - The organization of our code into subsystems
  - The use of proven best practices and patterns
- Well designed code:
  - Can be easily maintained and modified (It’s easier to understand)
  - Is more efficient and avoids technical debt
  - Deploys more effectively and easier into modern environments like Docker
  - Follow good engineering practices
  - Manages complexity efficiently

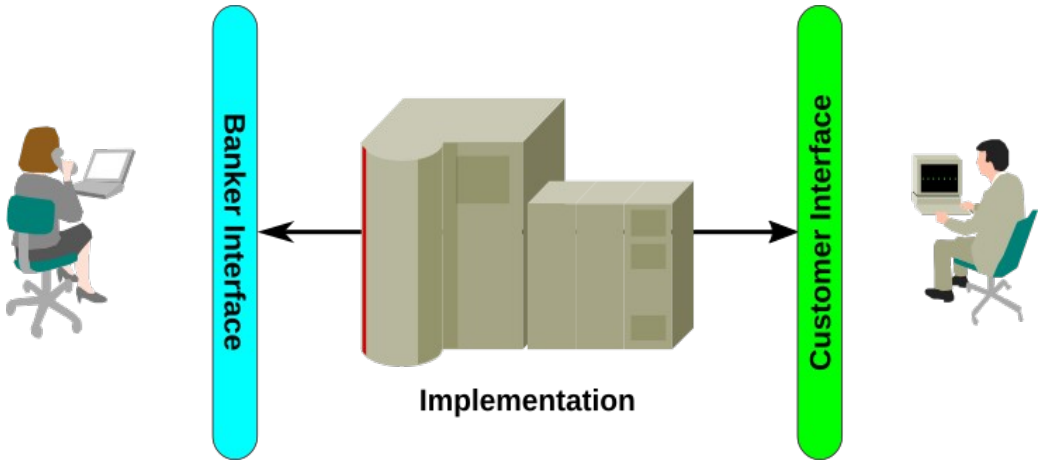
Object Oriented Fundamentals

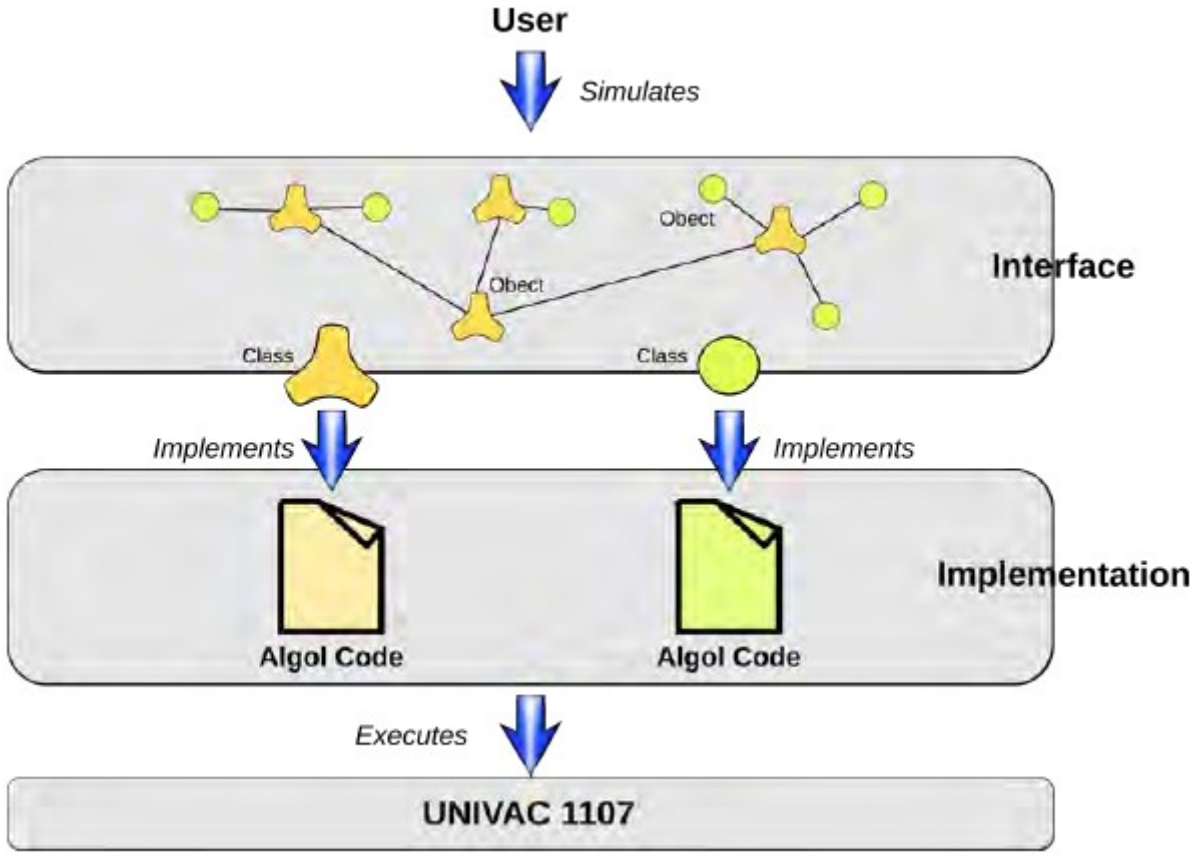
- OO is a software design paradigm
  - Defines specific techniques, tools and concepts for analyzing problems and developing solutions
  - Adheres to good engineering practices
- Originated in the 1960s
  - First OO programming language was Simula in 1961
  - Emphasized usability of interfaces
    - Decoupled from underlying design
    - Based on domain objects
  - Software should have iconic user interfaces
    - Requires an in depth understanding of what users know and do

"Why should people have to learn how to interact with a computer? Why can we not design a computer program that resembles what it automates so that people can interact with it based only on what they already know?"

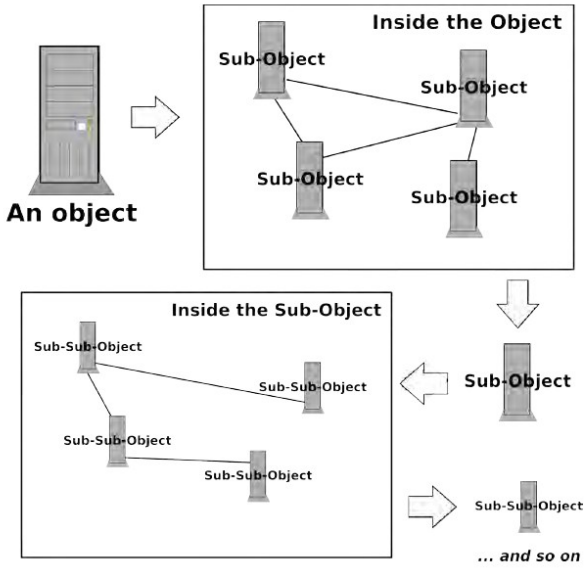
Professors Ole-Johan Dahl and Kristen Nygaard

Interface vs Implementation

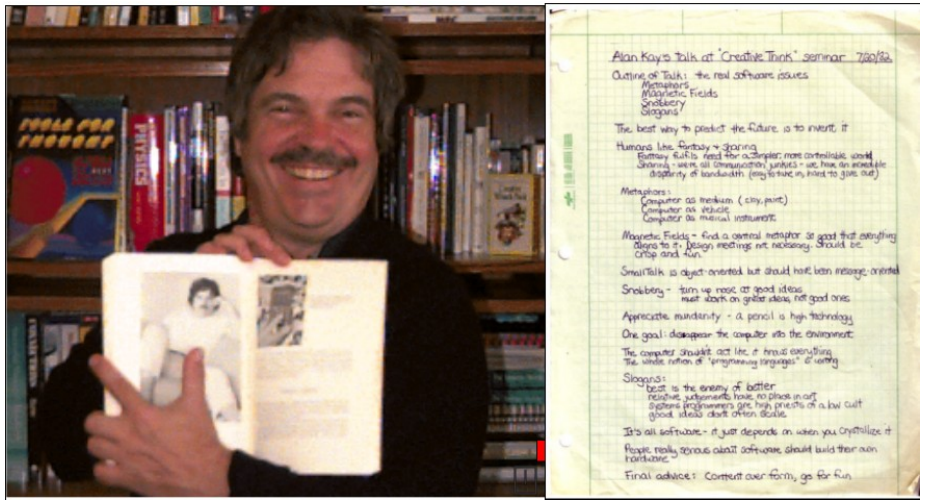




Object Oriented Programming



1. A system is built up in layers.
2. Each layer is itself an object.
3. Each layer is made up of a collection of peer objects which provide the functionality of that layer
4. There is no restriction to the types of objects that exist within a particular layer.



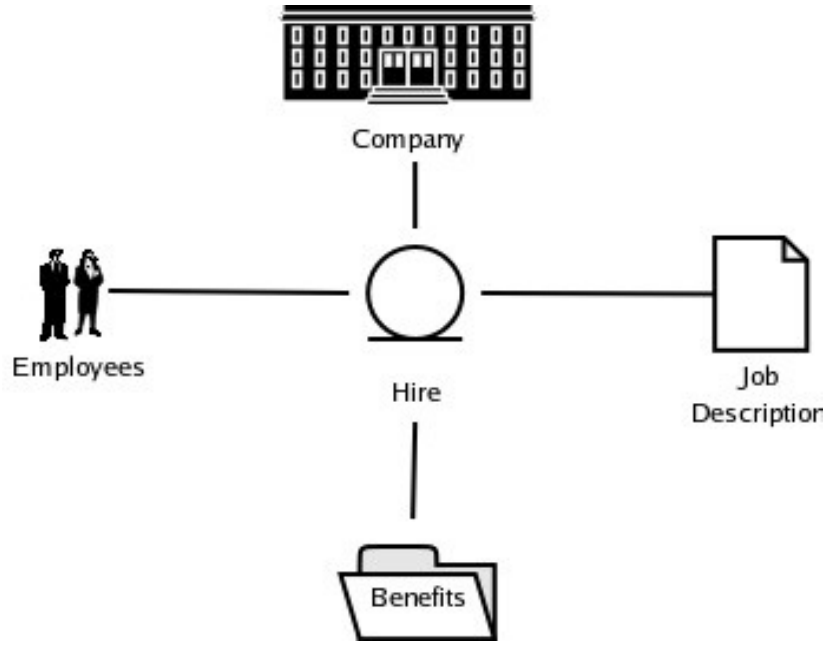
I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages (so messaging came at the very beginning – it took a while to see how to do messaging in a programming language efficiently enough to be useful)

Instead of dividing a system (or computer) up into functional subsystems, we think of it as a being made up of a collection of little processing engines, called objects. Each object will have similar computational power to the whole and processing happens when the objects work together. However, and this is the recursive part, each object can then be thought of in turn as a collection of sub-objects, each with similar computational power to the object, and so on.

Alan Kay

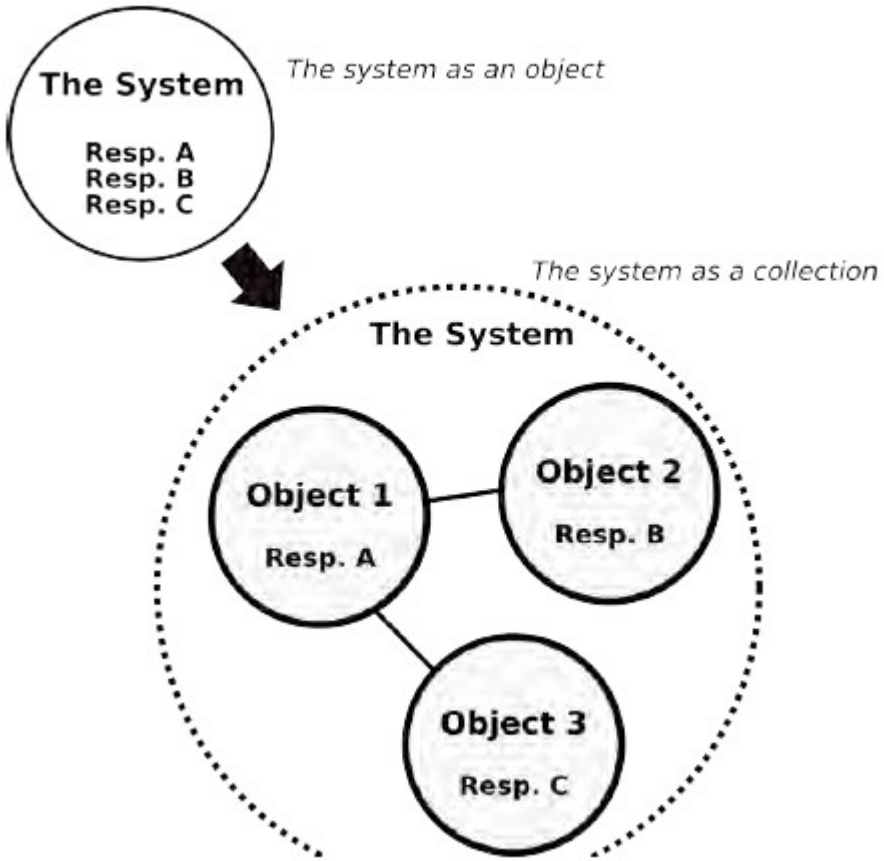
OO Programming Design Principles

- **One:** Everything can be modeled as an object.



OOP Design Principles

- **Two:** Systems are collections of objects collaborating for a purpose and coordinating their activities by sending messages to each other.
- **Three**: An object can have an internal structure composed of hierarchies of sub-objects.
- **Four**: Objects within a system have individual responsibilities, the responsibility of a system as a whole is distributed across the objects that make up the system by delegating specific responsibilities to individual objects.



OOP Design Principles

- **Five**: An object presents an interface that specifies which requests can be made of it and what the expected results of those requests are.
  - This interface is independent of the actual internal workings of the object – remember the separation of interface and implementation from before.
- **Six**: An object is of one or more types. A type includes a role the object plays, a set of responsibilities and an interface. Each type may be derived from a hierarchy of types.

SOLID Principles

- Engineering guidelines for developing OO code
  - Application of Engineering Principles
- Single Responsibility Principle (SRP)
- Open/Closed Principle
- Liskov's Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Interface Design

- Key to implementing the open-close principle
  - Two main types of interfaces
- Imperative
  - Programming API interface made up procedure calls
  - Clients use an interface method to request a specific action be performed
  - This is a synchronous interface since the client has to wait for the server to respond
  - Ideal for APIs where we want to call specific functionality from a program module
- Declarative
  - Provides an object and what should happen
  - Client does not specify how to do it – the receiver figures out how to process the object
  - Used by functional programming, messaging systems – command design pattern
  - Better alternative for applications that provide services
  - Commonly used to implement asynchronous applications

Accounting Example

- Assume we have an imperative interface to an accounting system
  - Each operation needs a specific request
- Violates the open close principle
  - Extending functionality requires changing the interface
  - This will break any clients that depend on the old interface
- We also have no sense of what the various return values mean.
- Equivalent to an accountant not doing anything without an explicit order
  - “This is a bill, pay it.”
  - “This is check, deposit it”

```
package acctdept;

public interface Accounting {
    int pay(Invoice invoice);
    int deposit(Check check);
    boolean payment(String vendor, int amount);

}

class Check {}
class Invoice{}
```

Remember This?

- A hospital provides a healthcare service
- Made up of individual microservices
  - Laboratory
  - X-Ray
  - Pharmacy
  - Medical Staffing
- Each microservice:
  - Specializes in a specific domain activity
  - Only that microservice performs that domain activity (the pharmacy doesn't do x-rays for example)
  - Each microservice operates autonomously

And This?

- Microservices request services from each other
  - They do not need to know the internal workings of the other microservice
- Requests are made through interfaces
  - Called APIs in software engineering
  - These are often paper based in the real world
  - Requisitions and official forms and paperwork used to make requests of a service
  - Response to a request is often a report of some kind
- Microservices make sense intuitively

COVID-19 VIRUS LABORATORY TEST REQUEST FORM[1]

Submitter information			
NAME OF SUBMITTING HOSPITAL, LABORATORY, or OTHER FACILITY*			
Physician			
Address			
Phone number			
Case definition:[2]	☐ Suspected case ☐ Probable case		
Patient info			
First name		Last name	
Patient ID number		Date of Birth	Age:
Address		Sex	☐ Male ☐ Female ☐ Unknown
Phone number			
Specimen information			
Type	☐ Nasopharyngeal and oropharyngeal swab ☐ Bronchoalveolar lavage ☐ Endotracheal aspirate ☐ Nasopharyngeal aspirate ☐ Nasal wash ☐ Sputum ☐ Lung tissue ☐ Serum ☐ Whole blood ☐ Urine ☐ Stool ☐ Other:		
All specimens collected should be regarded as potentially infectious and you must contact the reference laboratory before sending samples. All samples must be sent in accordance with category B transport requirements.			
Please tick the box if your clinical sample is post mortem ☐			
Date of collection		Time of collection	
Priority status			
Clinical details			
Date of symptom onset:			
Has the patient had a recent history of travelling to an affected area?	☐ Yes ☐ No	Country Return date	
Has the patient had contact with a confirmed case?		☐ Yes ☐ No ☐ Unknown ☐ Other exposure:	
Additional Comments			

Accounting Example

- More realistically
  - The accountant would find documents in their inbox
  - They would then do the appropriate type of processing depending on the type of the document
  - Once the processing is done, some sort of report is returned with information about what was done during the processing
- The accountant would probably refer the document to specific department for processing
  - For example, invoice go to Accounts Payable, checks go to Accounts Receivable
  - The accountant routes documents to the appropriate handler
  - Clients don’t need to know anything about where their request is being processed or how it is being processed.
  - This is an implementation of the façade design pattern


Accounting App
Demo

Why Do This?

- Designing applications like this takes more upfront work but:
  - It reflects the structure of the domain – it's connected to reality
  - It provides a common bases of understanding between the business and developers
  - It's easier to understand and maintain
- This sort of design is easily implementable as:
  - Synchronous microservices using REST
  - Asynchronous message based microservice
- It is consistent with best practices in design and development

Domain Driven Design

Lab Design 1

Specifications

- Before we build anything, we develop a specification
  - This describes exactly what we are building before we design or build it
  - This is a standard engineering practice
- When using Agile
  - The specification evolves as we discover more requirements in each specification
  - Traditionally, Agile developers use a set of acceptance tests for the system as a specification
- A gold standard is the IEEE *Best Practices for Software Requirements Specifications*

Features of a Good Spec

Software Quality Engineering Checklist

Complete	*The specfication covers all possible inputs and conditions, both valid and expected as well as invalid and unexpected..*
Consistent	*No two specification items require to system to behave differently under the same conditions, inputs and system state.*
Correct	*The expected result of each test meets the is consistent with the appropriate business logic, process and documentation.*
Testable	*Each specification item is quantified and measurable so that a pass/fail test can be written for it..*
Verifiable	*There is a finite cost effective process for ensuring each specification item is in the final product and can be tested.*
Unambiguous	*There is only possible interpretation of each specification item.*
Valid	*All stakeholders can read, understand each specification item and has formally approve the item.*
Modifiable	*The specification itemsare organized in a way so that they are easy to use, modify and update.*
Ranked	*The specification items are in a priority order that everyone on both the and technical sides agree on.*
Traceable	*Every specification item can be traced back to the original requirements criteria that it is intended to satisfy.*

Architecture

Lab Design 2

Robustness Analysis

- Most specs are developed under the tacit exception that they meet all of the properties of a good spec.
- The purpose of a robustness analysis is to:
  - Analyze the design and specification to look for errors, omissions and inconsistencies
  - We use the general properties of completeness, correctness, etc.
  - Develop a set of acceptance tests that fully test that the system will work as specified
  - To look for technical issues that may affect the system (server crashes, data transmission errors for example)

Ivar Jacobson introduced the concept of robustness analysis to the world of OO in 1991. It involves analyzing the narrative text of your use cases and other requirements and synthesizing the requirements models into a single conceptual specification model.

Doug Rosenberg

Robustness Analysis

- Rosenberg states that robustness analysis ensures that the inputs from the following four sources are all integrated and checked for robustness
  - The functionality described in the business or other processes requested or needed by the various stakeholders
  - The domain objects identified by the users and the use cases (specifically we need them to be described sufficiently for iconic interface design and database definitions)
  - Stakeholder and user functional requirements
  - System constraints and factors that will constrain our eventual design

Robustness Analysis

- In robustness analysis we critically examine the specification that is being developed with the following checks being performed:
  - **Sanity check**: make sure that the specification is correct and that doesn't require unreasonable or impossible system behaviour given the constraints that we have to work with
  - **Completeness check:** make sure that the requirements address all the necessary alternate courses of action and that all of the stakeholder requirements are fulfilled
  - **Consistency Check:** make sure that the various parts of the specification are not contradictory and that the same requirement is not fulfilled differently in different parts of the system
  - **Exception Check:** make sure that the system recovers appropriately from all the abuse cases and exceptions identified in the business processes and requirements,including time-out conditions
- If we allow these kinds of problems our specification, we will eventually wind up doing rework
  - A robustness analysis ensures that we find these problems before we commit resources to the development of the specification

Accounting Robustness Analysis

Demo

Robustness Analysis

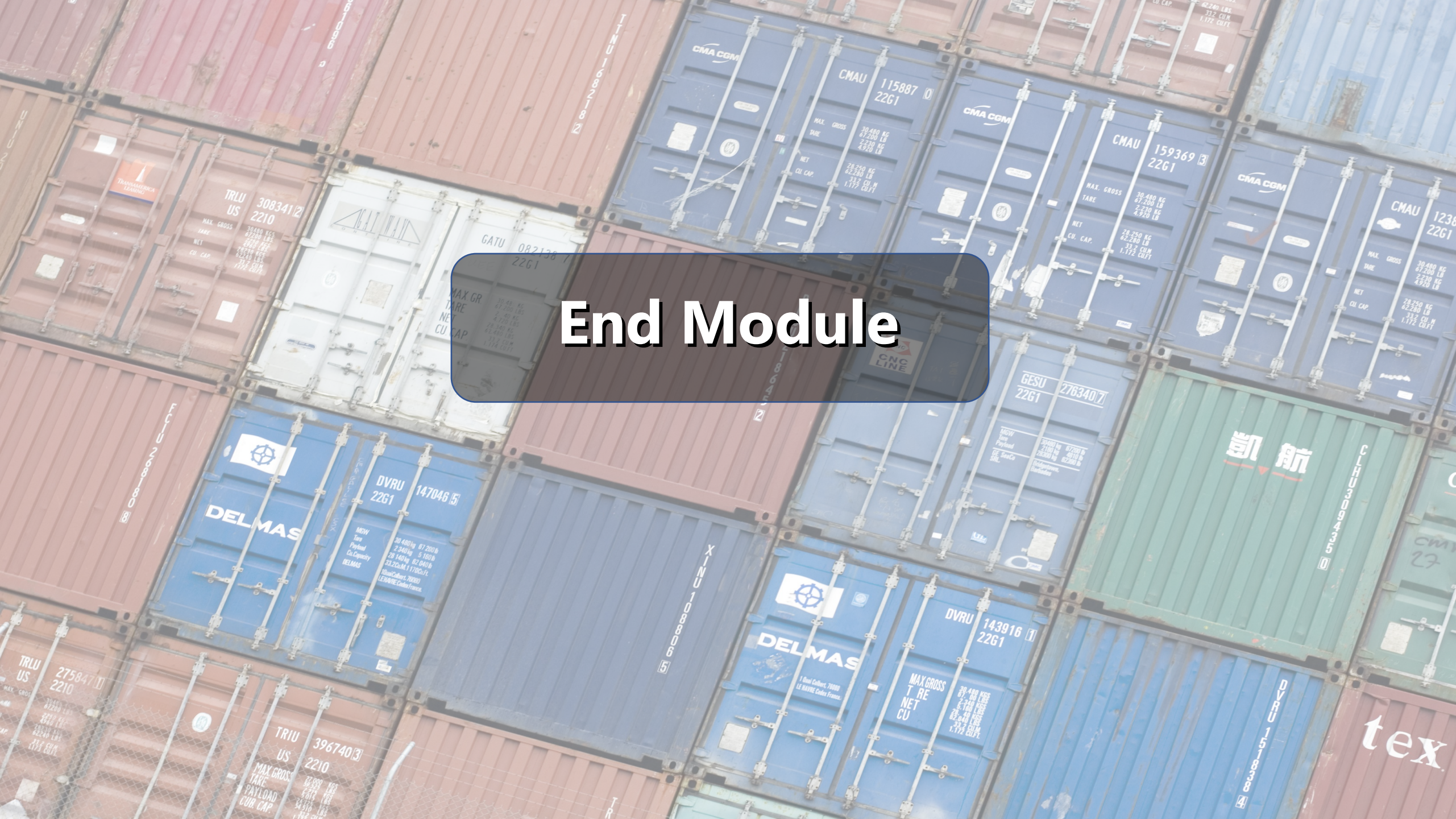
Design Lab 3

Working with Docker Images

Demo

Class Project Discussion


Questions?


End Module