Design Patterns: An Introduction

CS6.401 Software Engineering

Dr. Karthik Vaidhyanthan

karthik.vaidhyanathan@iiit.ac.in

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Acknowledgements

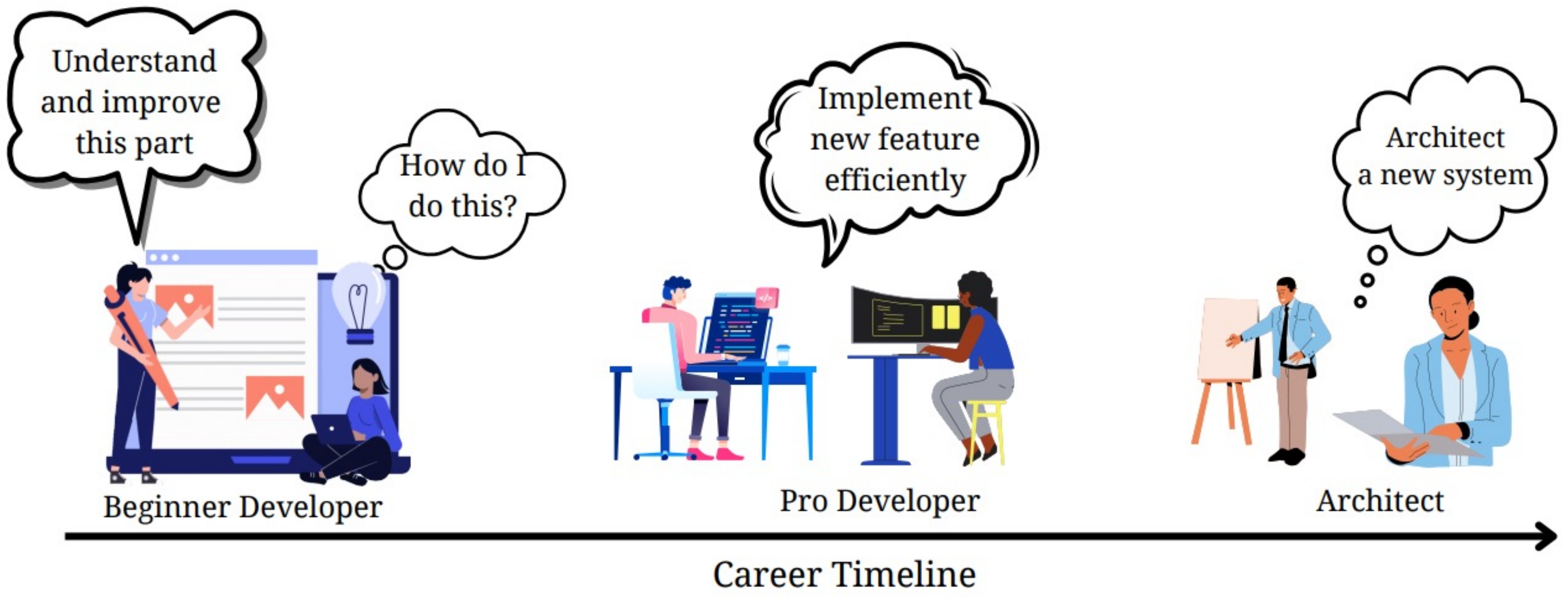
The materials used in this presentation have been gathered/adapted/generated from various sources as well as based on my own experiences and knowledge

-- Karthik Vaidhyanathan

Sources:

1. **Design Patterns: Elements of Reusable Object-Oriented Software** by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
2. Applying UML and Patterns, Craig Larman

The Journey



This Course

Modeling	Design Principles	Architectural Framework
Refactoring	Design Patterns	Architectural Patterns

What were some Lessons Learned form Unit 1?

Key Design Principles

- Abstraction
- Encapsulation
- Modularization
- Hierarchy

So all we need to follow them – Problem Solved!!

Designing that too OO Systems is not very straightforward

How to identify objects

How to group objects to classes

Interfaces have to defined

Hierarchies too!

Relationships among Objects!!

Things Improve with Practice

- Designs should be reusable, flexible and understandable
- Very difficult to get it right the first time – Not hard though!!
- Experience people also take multiple iterations
- Novice find it even more difficult to get their head around!

Experts are able to make good design….How?

Things Improve with Practice

- Experts tend to reuse solution that have worked in the past!
- The way objects are identified, relationships are established becomes recurring activity
- When something has been tried and worked well, why not use it again!!
- They start seeing recurring **patterns** over time
- What if this experience could be recorded for reuse?

GRASP

General Responsibility Assignment Software Patterns or Principles

- **Information Expert:** Who gets the responsibility?
  - Find which class has the data
  - The one who has data also should have the operations to perform the data

- **Creator:** Who gets the role of the creator?
  - Defines guidelines for which class should be in charge of creating objects of other type
  - E.g. Class B should be in charge of creating objects of A if:
    - B contains or compositely aggregates A
    - B closely uses A
    - B has inputs to construct A
    - B records A

General Responsibility Assignment Software Patterns or Principles

- **Low Coupling:** How to minimize impact of change?
  - Assign responsibilities such that to reduce coupling
  - Given two alternatives, chose the one that minimizes coupling

- **High Cohesion:** How to keep everything together in one object to better manage and to minimize coupling?
  - Do one thing and do it very well
  - Give one end-to-end responsibility to one class
  - Reduce communication

General Responsibility Assignment Software Patterns or Principles

- **Protected Variation:** How to protect part of a class from changes in part of another class?
  - Related to ensuring low coupling
  - Code of a part of class B is protected from changes in code of part A
  - Introduce interface around the unstable part of the codebase

- **Indirection :** How to ensure that one can communicate with another without knowing each other well?
  - Another principle/pattern to reduce coupling
  - Introduce a new class between two classes A and B
  - Changes in A or B doesn't affect each other. The intermediary absorbs the impact
  - Introduces a class as opposed to protected variation

General Responsibility Assignment Software Patterns or Principles

- **Polymorphism : How to decouple clients from different ways of accomplishing a single task?**
  - Contributes to low coupling
  - Several ways to accomplish a task or a functionality
  - Achieved through interfaces, overloading methods of super classes

- **Pure Fabrication : Whom to assign the responsibility when it does not fit into either of the classes?**
  - Promotes cohesion
  - Sometimes a responsibility needs to be assigned but need not fit well into a class
  - Create a new class (does not map to domain object for handling the responsibility

General Responsibility Assignment Software Patterns or Principles

- **Controller: What if there is a need for someone to control the responsibility between classes?**
  - Kind of a subtype of pure fabrication
  - Very common in UI applications -> between UI and the backend
  - Separate concerns clearly between two classes by having someone in middle
  - Does not map to any domain object

Design Patterns

Design Patterns

Each Pattern describes a problem which ***occurs over and over again*** *in our* ***environment*** *and then* ***describes the core of the solution*** *to that problem, in such a way that you can* ***use this solution a million times over****, without ever doing it the same way twice*

-- Christopher Alexander

Patterns captures {Context, Problem, Solution}

What are some of the patterns you can think of? 🤔

Patterns patterns everywhere!

- We have a natural tendency to look for patterns in anything and everything
  - Pattern of grades for courses
  - Pattern of questions in question papers
  - Climate patterns (rainfall, summer, …)
  - …

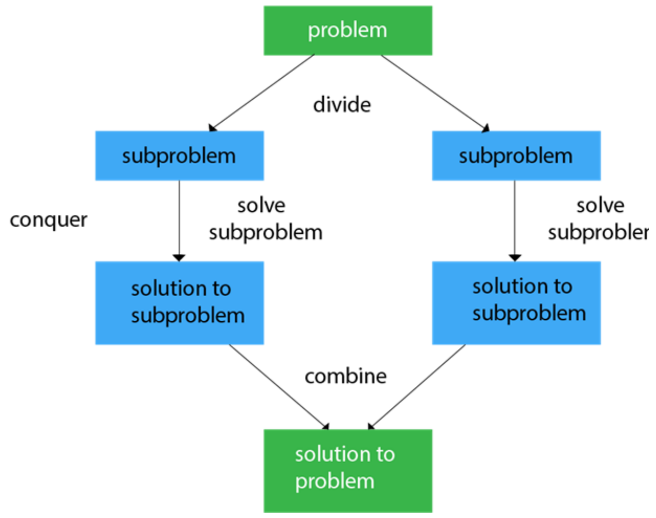
Architectural Patterns

Roman architecture

Color Patterns

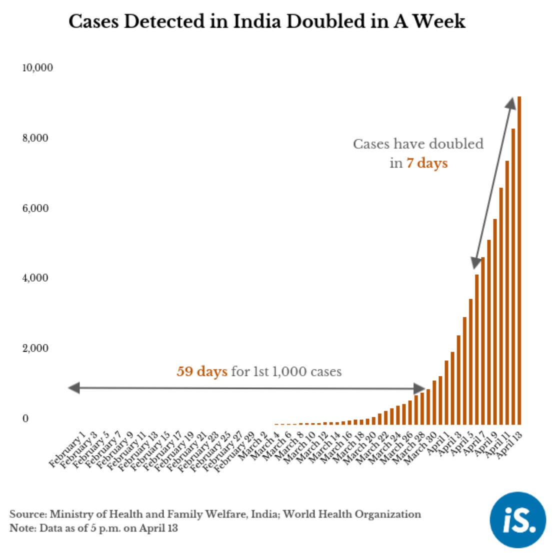
Island houses in Greece

Algorithmic Patterns



Divide and conquer

Data Patterns

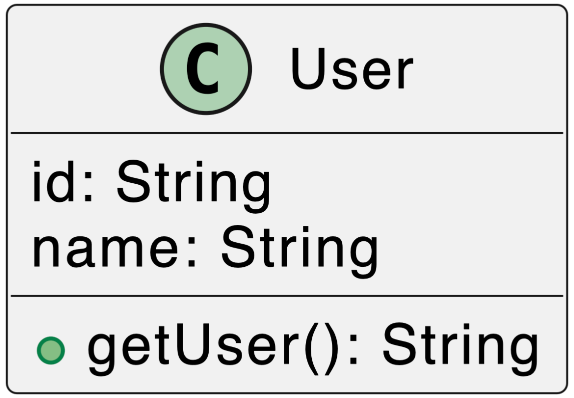


Covid cases curve

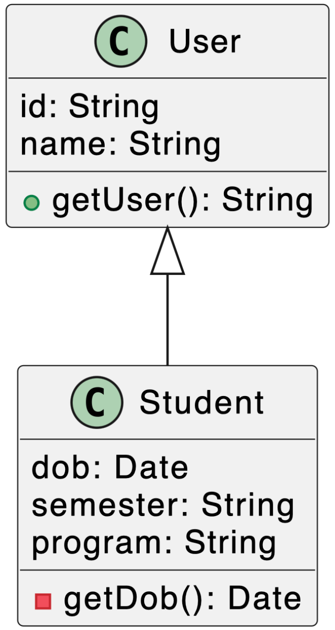
What about Software?

Many patterns to design and build software systems

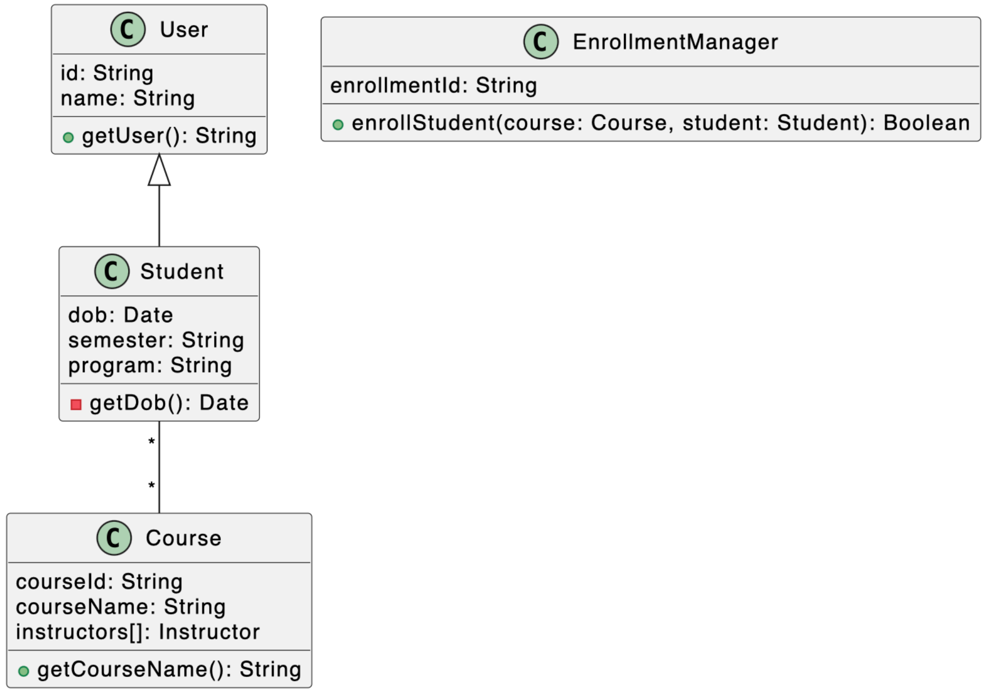
- Architectural Patterns [Higher Level]
- Design Patterns [Lower level]



Patterns for extracting objects
And classes
(Look for nouns, verbs, etc.)



Patterns for structuring everything



Patterns for distributing functionality

Four Elements of a Pattern

- **Pattern Name:** Handle to describe a design problem
- **Problem:** When to apply the pattern, preconditions, special relationships, etc.
- **Solution:** Elements that make up the design, relationships and collaborations
  - Not a particular solution but abstract representation with potentials
- **Consequences:** Results and trade-off of applying a given pattern
  - Perform cost-benefit analysis

Design Patterns

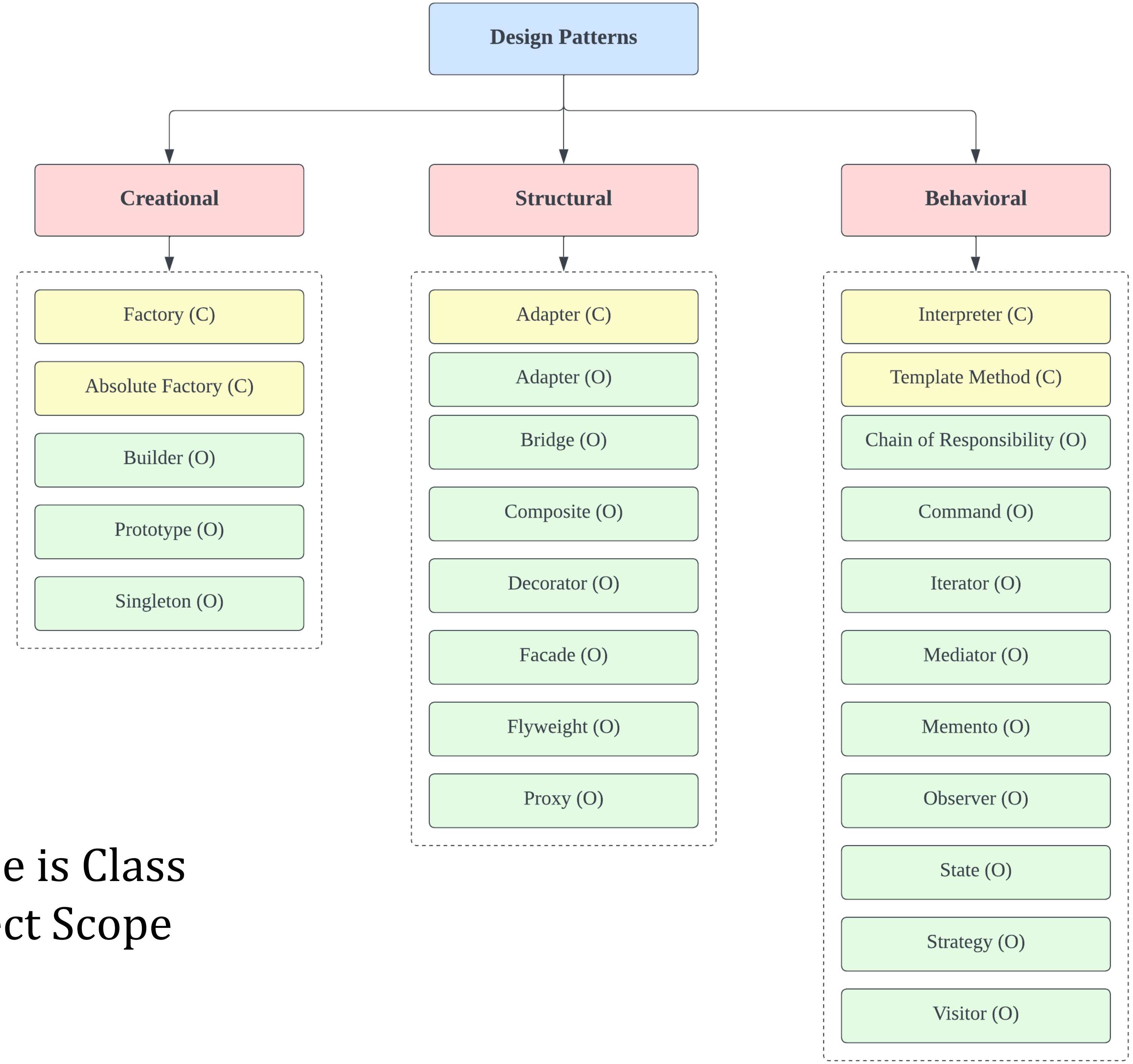
- Principles, relationships and techniques for creating **reusable** OO design
- Identifies participating objects, their roles, responsibilities and relationships
- **Not about** Linked Lists, hash tables, etc.
  - The are low level structures inside classes
- **Not about** complex domain specific design or design of subsystems
  - Domain specific design is more at high level – Architectural level

Classification of Design Patterns

- Mainly divided into three based on the purpose they serve
- Creational, Structural and Behavioral
- Each category has a purpose, a set of patterns that work in different scope:
  - Class or object
- There are a total of 23 classic patterns: Gang of Four (GOF) patterns
  - The famous book *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

Classification of Design Patterns

- Creational
  - Class - Defer creation to subclasses
  - Object – Defer creation to another object
- Structural
  - Class – Structure via inheritance
  - Object – Structure via Composition
- Behavioral
  - Class – algorithms/control via inheritance
  - Object – algorithms/control via object groups



C – Scope is Class
O – Object Scope

Describing Patterns

- Pattern Name and Classification
  - Name captures essence and classification the category it tackles

- Intent
  - What does the design pattern do?
  - What is its rationale and intent – What problem does it address?

- AKA (Also Known As): Other known names

- Motivation
  - A scenario that illustrates the problem and how pattern can solve it

- Applicability
  - What are the situation in which the pattern can be applied and how to recognize them?

Describing Patterns

- Structure
  - Graphical representation of the pattern in UML or other modeling language
- Participants
  - The classes/objects participating and their responsibilities
- Collaborations
  - How the participants collaborate to carry out their responsibilities.
- Consequences
  - How well does the pattern support its objectives?
  - What are the trade-offs and results of using the pattern?
  - What part can be varied independently?

Describing Patterns

- Implementations and Sample Code
    - Code fragments to illustrate implementation in OOP language of choice
- Known Uses
    - Examples of patterns in real systems
- Related Patterns
    - What are the patterns closely related to this one?
    - What are the key differences?
    - What other patterns with which this can be used?

Some Principles

Program to Interface Not Implementation

- One of the most important OO Design Principles
- “Program to interface” refers to the idea of ensuring loose coupling
  - Does not only mean the “Interface”?
- Very useful when lot of changes are expected
- Create an interface, define methods -> create classes that implements them
- Allows external objects to easily communicate
- Maintainability and flexibility increases

Favor Object Composition over Class Inheritance

- Two most common techniques: Inheritance and Composition

- Class inheritance: White-box reuse
  - Internals of parent class are visible to child class
  - Defined statically at compile time
  - Sub class can override methods of parent class

- Inheritance is not always the go to solution - ”breaks encapsulation”

- Composition: Black-box reuse
  - Objects acquiring references to other objects
  - Defined dynamically at run time
  - Encapsulation is not broken – Objects are accessed through interfaces
  - Get what is needed by assembling and not by creating

Thank You

Course website: karthikv1392.github.io/cs6401_se

Email: karthik.vaidhyanathan@iiit.ac.in
Web: https://karthikvaidhyanathan.com
Twitter: @karthi_ishere

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Software Engineering Research Centre