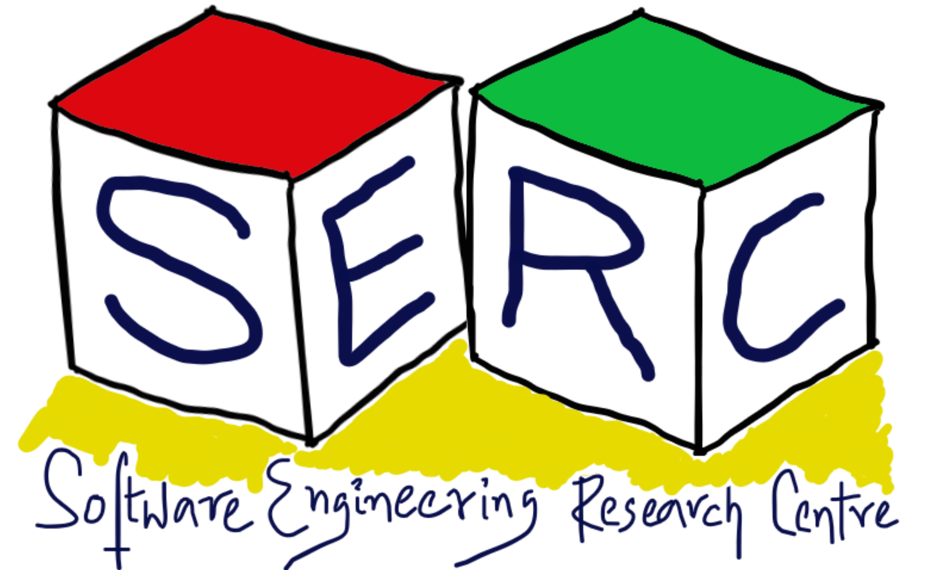
Design Patterns

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

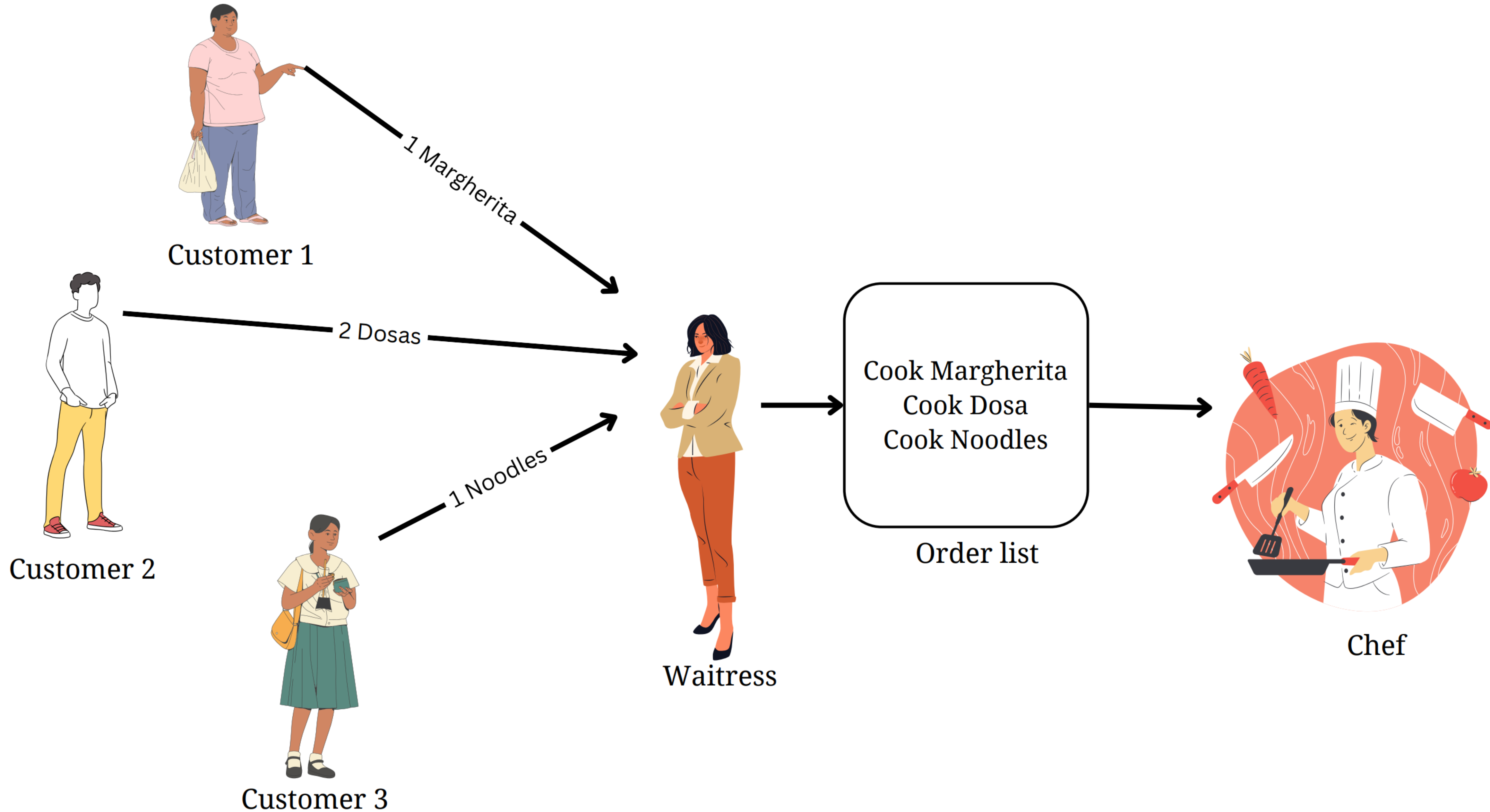
-- Karthik Vaidhyanathan

Sources:

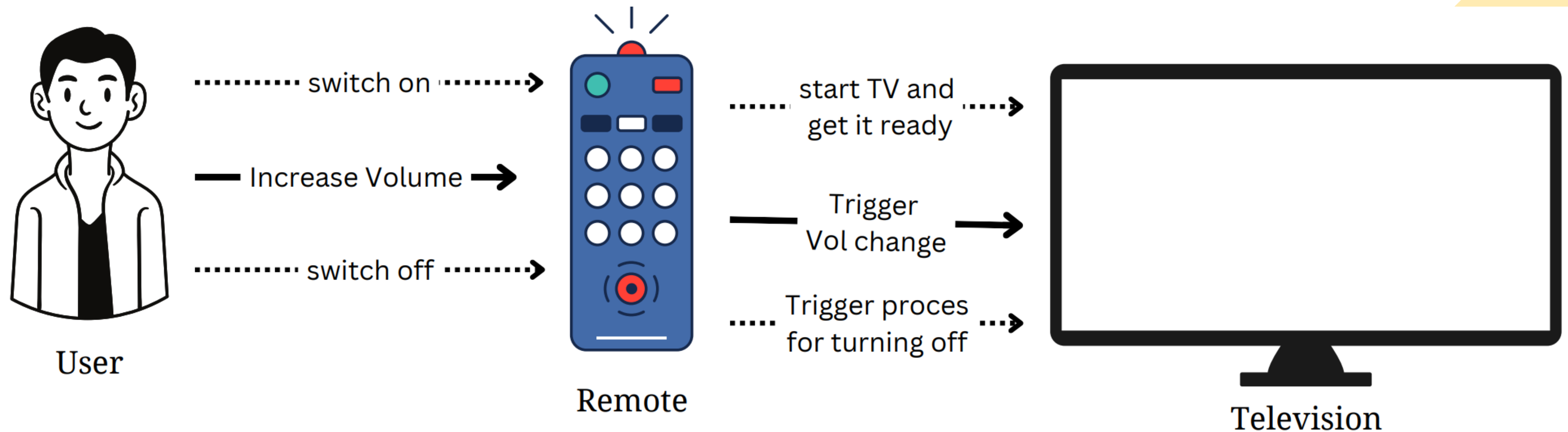
1. **Design Patterns: Elements of Reusable Object-Oriented Software** by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
2. **Head first Design Patterns**, Second Edition, Eric Freeman and Elisabeth Robson

You can give a command:
Command Pattern
[Behavioral]

Meet the Command Pattern!



Meet the Command Pattern – A Scenario



Should remote know exactly how the TV work step by step?

SERC
Software Engineering Research Centre

Meet the Command Pattern

- What if sender need not have to worry about receiver's internal implementation?
- What if some commands needs to be scheduled and executed in order at a later time?
- Sender needs to be decoupled from a receiver
- Encapsulates everything required to perform an action
  - Execution of action can happen independently

SERC Software Engineering Research Centre

Command Pattern: Documentation

Intent

Encapsulate a request as an object, allowing parameterization of clients with requests, log or queue request and support undoable operations.

Also Known As: Action, Transaction

Motivation

- Sometimes its necessary to request to objects without details about operation
- Objects can be stored and passed around -
- Five key objects: *Client, Command, Concrete Command, Invoker and Receiver*

Example: UI kits [Think about if you want to develop a button class]

SERC
Software Engineering Research Centre

Command Pattern: Documentation

Applicability

- Parameterize objects by an action to perform – Callbacks in procedural
- Specify, queue, execute request at different times
- Support undo operations – Think of editors, games [Add another operation in command interface]
- Support logging changes – Manage crashes
- Sometimes an operation may be composed of primitive operations

SERC
Software Engineering Research Centre

Command Pattern: Documentation

Structure

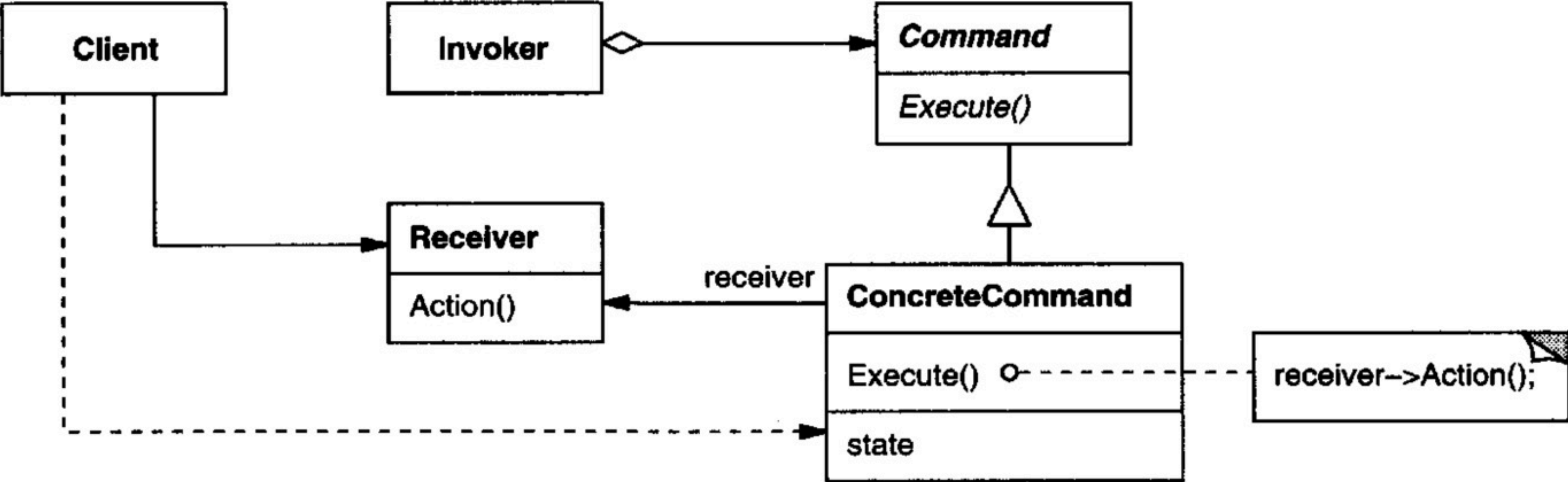


Image source: Gang of four book

Command Pattern: Documentation

Participants

Command (Command.java)

- Interface for executing an operation

ConcreteCommand (TVOnCommand, TVOffCommand,..)

- Binding between receiver object and action
- Implements the execute by invoking operations on receiver

Receiver (Television)

- Knows how to perform the operations associated with a request

Client (RemoteControlDemo)

- Create ConcreteCommand object and sets its receiver

Invoker (RemoteControl)

- Calls command to execute a request

This Photo by Unknown Author is licensed under CC BY

Command Pattern: Documentation

Consequences

- Decoupling client and receiver
  - Decouples invoke operation from the one that knows how to perform it
- Commands as first-class objects
  - Command can be manipulated and extended like any other object
- Composite commands can be formed
  - Commands can be composed to form a larger command
- Code complexity may increase
  - Not every time this is needed
  - Introduction of new layer between senders and receivers

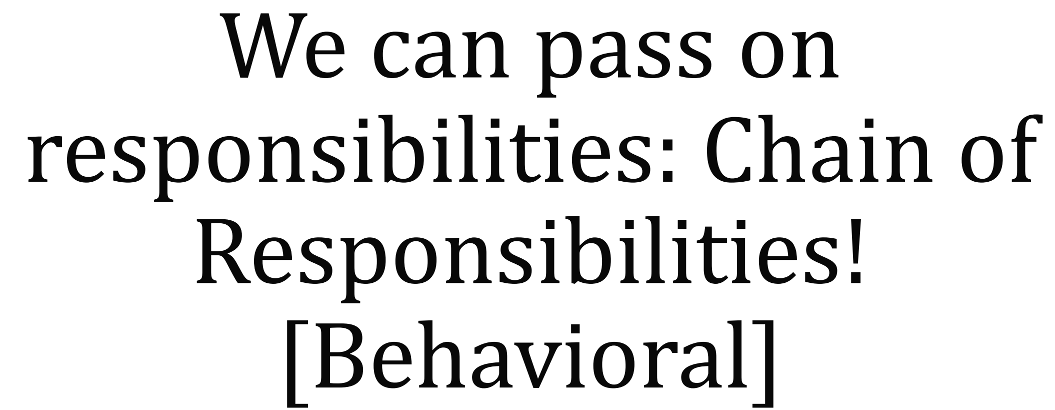
SERC
Software Engineering Research Centre

Command Pattern: Documentation

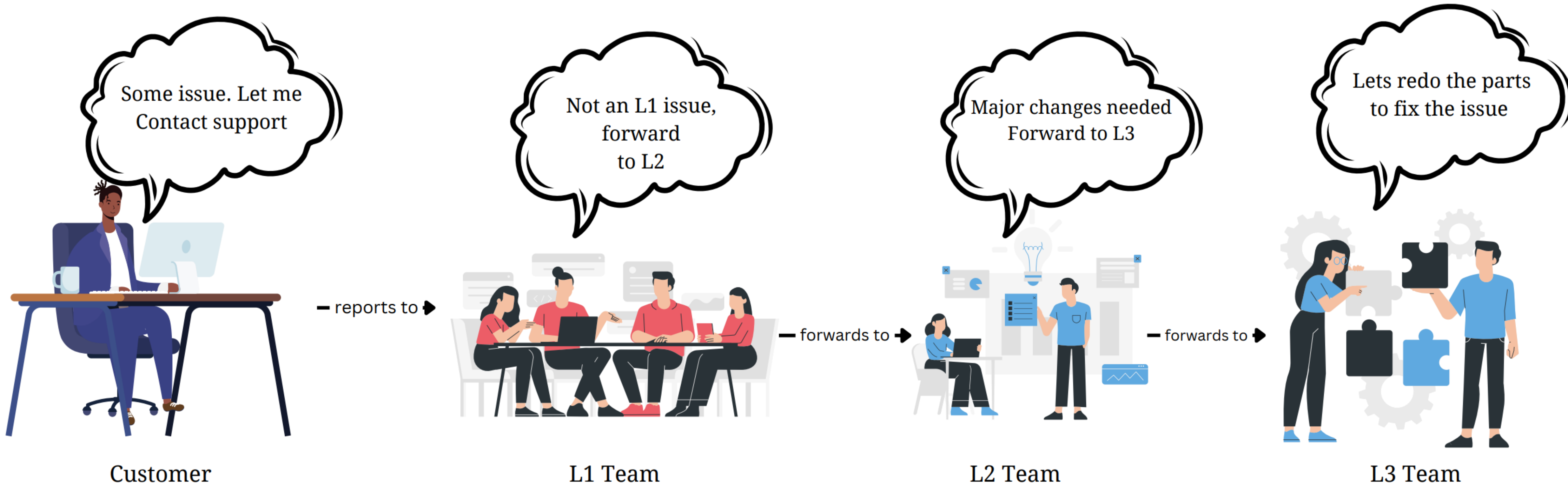
Implementation

Check the source code given along: RemoteControlCommand

SERC
Software Engineering Research Centre


We can pass on responsibilities: Chain of Responsibilities! [Behavioral]

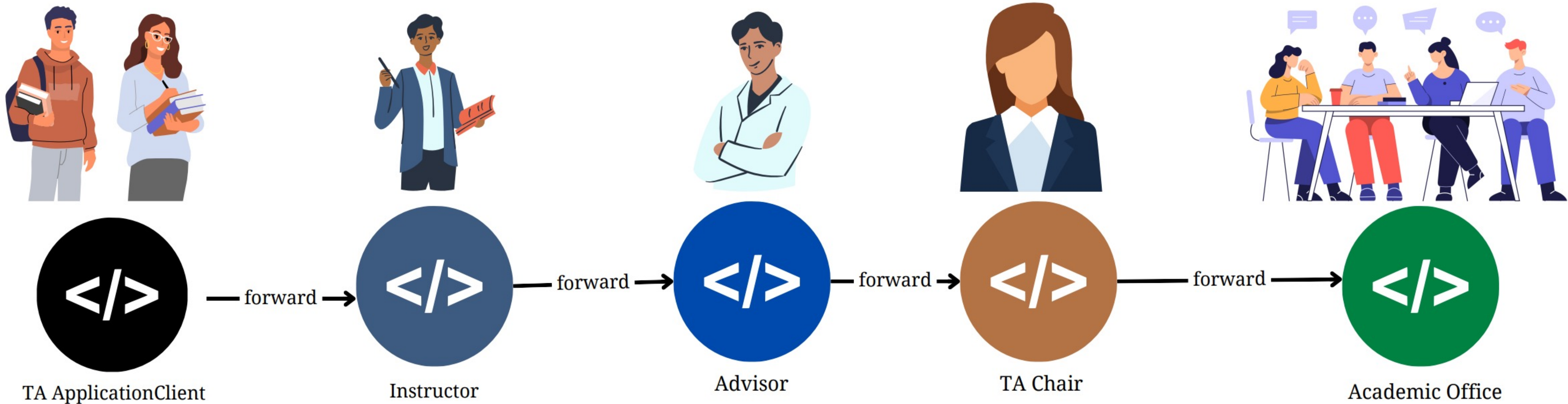
Meet the Chain of Responsibility Pattern!



SERC
Software Engineering Research Centre

Meet the Chain of Responsibility Pattern - Motivation

TA Application Scenario



How do you implement this ?

Software Engineering Research Centre

Meet the Chain of Responsibility Pattern

- What if one single request requires processing by multiple objects?
- What if the sender needs to be decoupled from receiver in the form of set of intermediatory objects?
- Sometimes single task may require multiple steps to process
- Each step in the process may decide if it needs to be further processed or not

SERC Software Engineering Research Centre

Chain of Responsibility Pattern: Documentation

Intent

Avoid coupling the sender of a request to its receiver by giving more than one object a chance top handle the request. Chain the receiving objects and pass the request along the chain until one handles it

Also Known As: CoR, Chain of Command

Motivation

- Request may have to be passed along a chain
- Senders and receivers need decoupling
- Key objects: *Handler, ConcereteHandler* and *Client*

Example: Payment process in an e-commerce system

SERC
Software Engineering Research Centre

CoR Pattern: Documentation

Applicability

- More than one object may handle a request and handler isn't known apriori
- Issue request to one object without specifying the receiver
- The set of objects that can handle a request should be specified dynamically

SERC Software Engineering Research Centre

CoR Pattern: Documentation

Structure

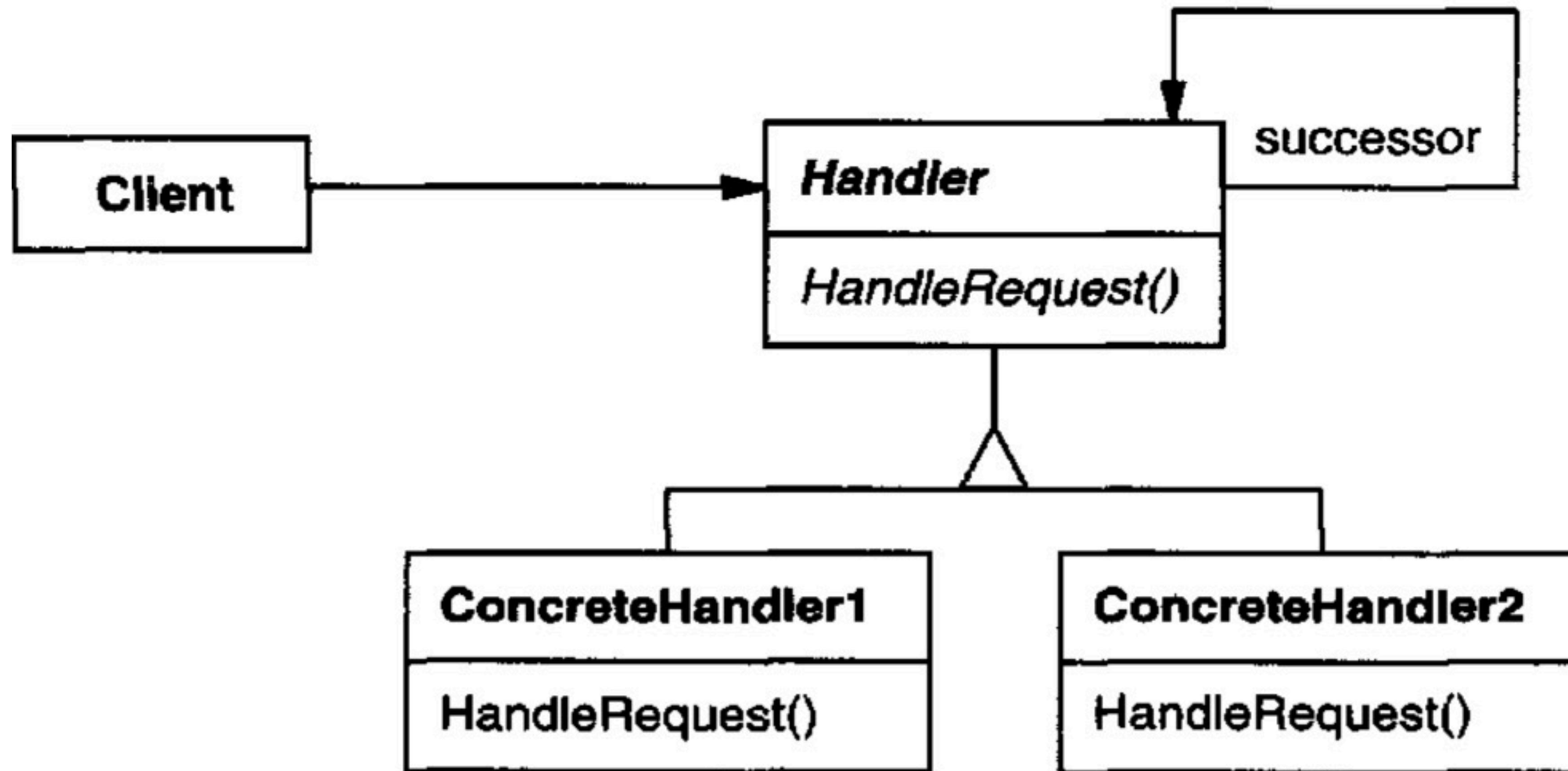


Image source: Gang of four book

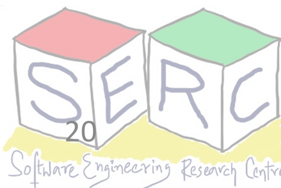
CoR Pattern: Documentation

Participants

Handler (ApplicationHandler)
- Defines an interface for handling requests

ConcreteHandler (InstructorHandler)
- Handles requests its responsible for
- Can access its successor

Client (StudentDemo)
- Initiates the request to a ConcreteHandler object on the chain



This Photo by Unknown Author is licensed under CC BY

CoR Pattern: Documentation

Consequences

- Reduced Coupling
  - Object does not need to worry about which other object handles request
  - Simplifies object interactions

- Flexible assignment of responsibilities
  - Flexible distribution of responsibilities among objects
  - Responsibilities of each handler can be changed at run time (chain can be increased)

- Receipt isn't guaranteed
  - Request has no explicit receiver – No guarantee of handling
  - Request can go unhandled when chain is not configured properly

SERC
Software Engineering Research Centre

CoR Pattern: Documentation

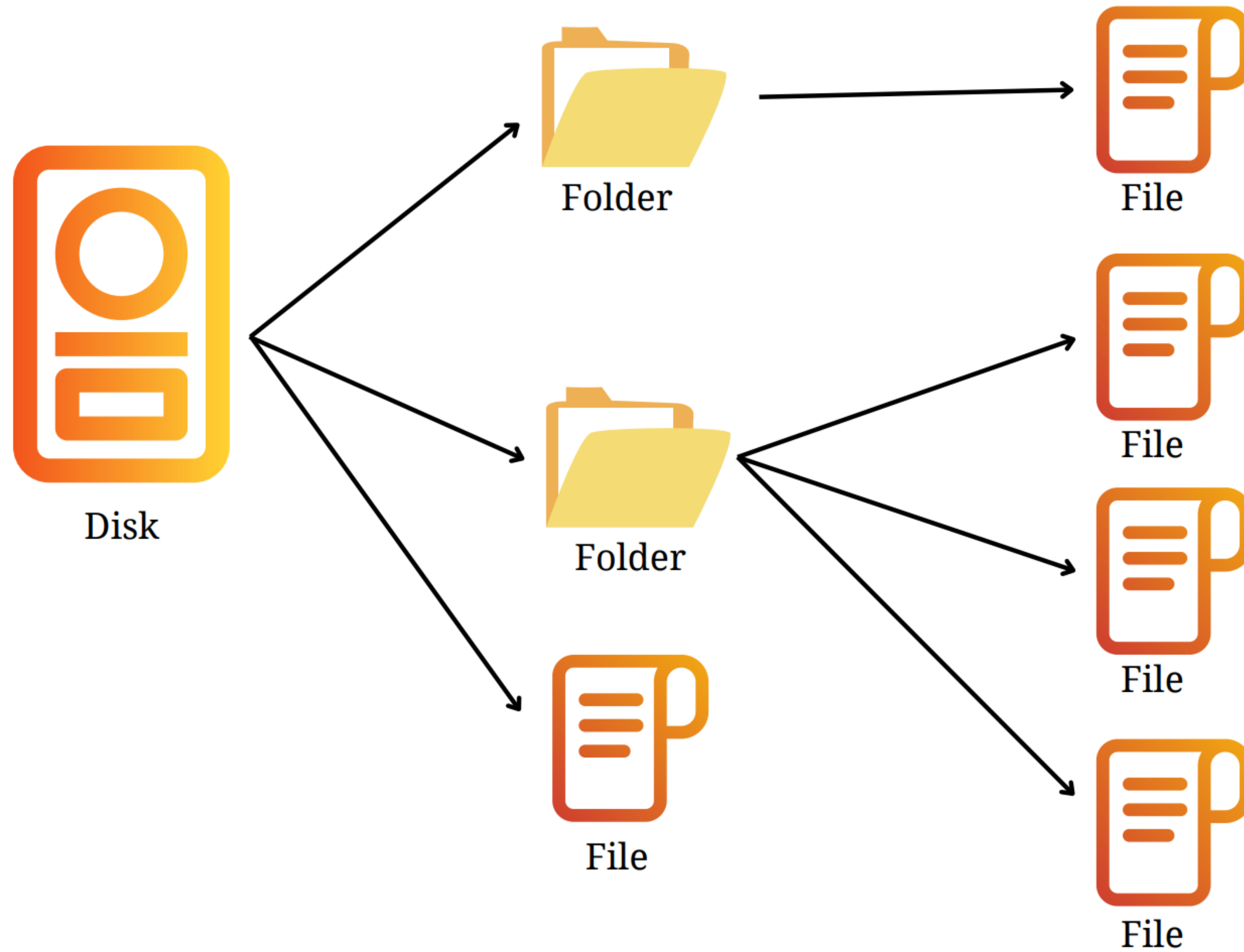
Implementation

Check the source code given along: TA-ApprovalChainOfResponsibility

SERC
Software Engineering Research Centre

How about Composing!:
Meet the Composite Pattern
[Structural]

Meet the Composite Pattern!

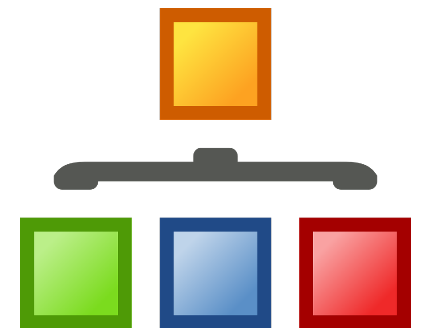


How to get the total size?

SERC
Software Engineering Research Centre

Meet the Composite Pattern

- What if a large component is composed of smaller components?
- What if the client need not worry about the complex hierarchy?
- What if the composition tree needs to be parsed recursively?
- Composition may contain primitives and larger components
- Have everyone in the tree share some common method



SERC
Software Engineering Research Centre

Composite Pattern: Documentation

Intent

Compose objects into tree structures to represent part-whole hierarchies. Composite lets client treat objects and compositions uniformly

Also Known As: Object Tree

Motivation

- Enable client to treat primitives and containers identically
- Promotes extensibility – Introduce new types
- Four key objects: *Component, leaf, composite and client*

Example: Disk system has folders and files. Each folder has files

SERC
Software Engineering Research Centre

Composite Pattern: Documentation

Applicability

- Represent part-whole hierarchies of objects
  - Recurse through the tree in a more controlled manner
- Clients should be unaware of the differences
  - Ignore difference between composition of objects and individual objects
  - All objects in the composite structure are treated uniformly

Composite Pattern: Documentation

Structure

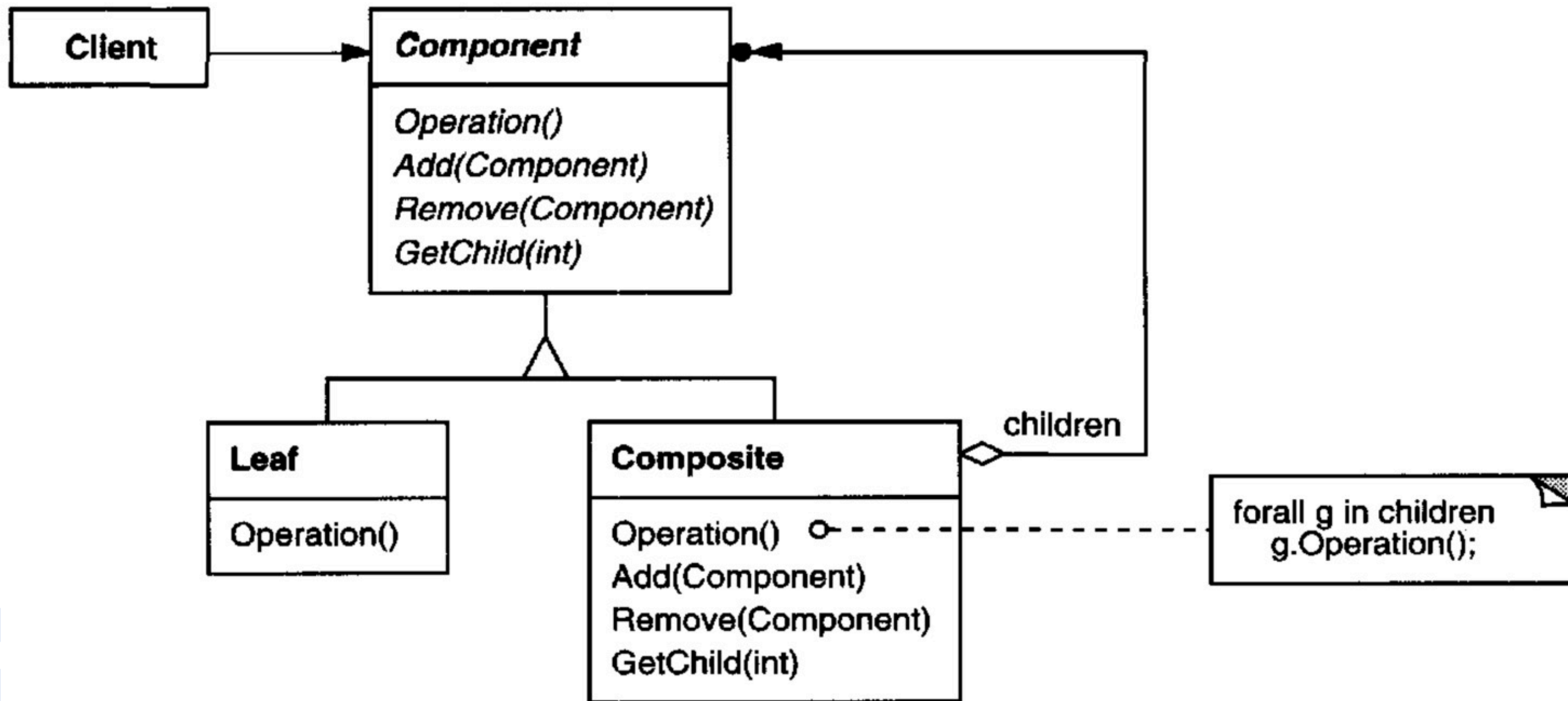


Image source: Gang of four book

Software Engineering Research Centre

Composite Pattern: Documentation

Participants

Component (FileSystem)

- Declares the interface for objects in the composition
- Implements default behavior – also declares interface to access child components

Leaf (File)

- Leaf objects in the composition – No children
- Behaviour of primitive objects is defined

Composite (Folder)

- Defines behavior for components with children
- Stores child components

Client (FileSystemDemo)

- Manipulates objects in the composition through component interface

This Photo by Unknown Author is licensed under CC BY

SERC
Software Engineering Research Centre

Composite Pattern: Documentation

Consequences

- Class hierarchies with primitive and composite objects
  - Primitive objects can be further composed
  - Client can work with both primitive and composite in same way
- Enhance client-side simplicity
  - Clients are not aware if an object is primitive or composite
  - No case statement or if conditions needed
- Add new components easily
  - New composite or sub-classes can be added to tree without affecting client
- Design can be too general – Also it sometimes can be forcefit
  - Restricting components of a composite is hard

SERC
Software Engineering Research Centre

Composite Pattern: Documentation

Implementation

Check the source code given along: FileSystemComposite

Software Engineering Research Centre

Thank You

Course website: karthikv1392.github.io/cs6401_se

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