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

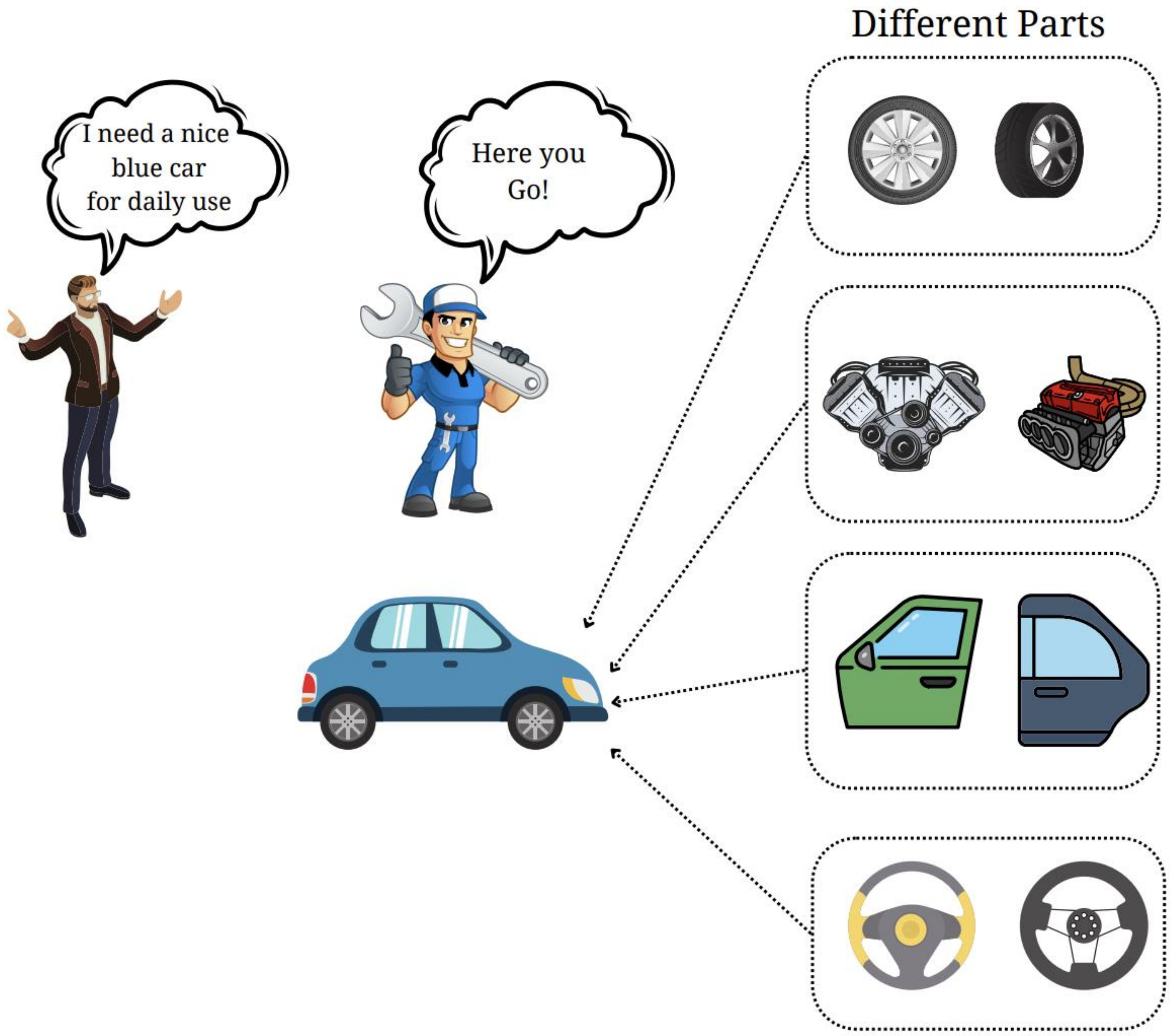
-- Karthik Vaidhyanathan

Sources:

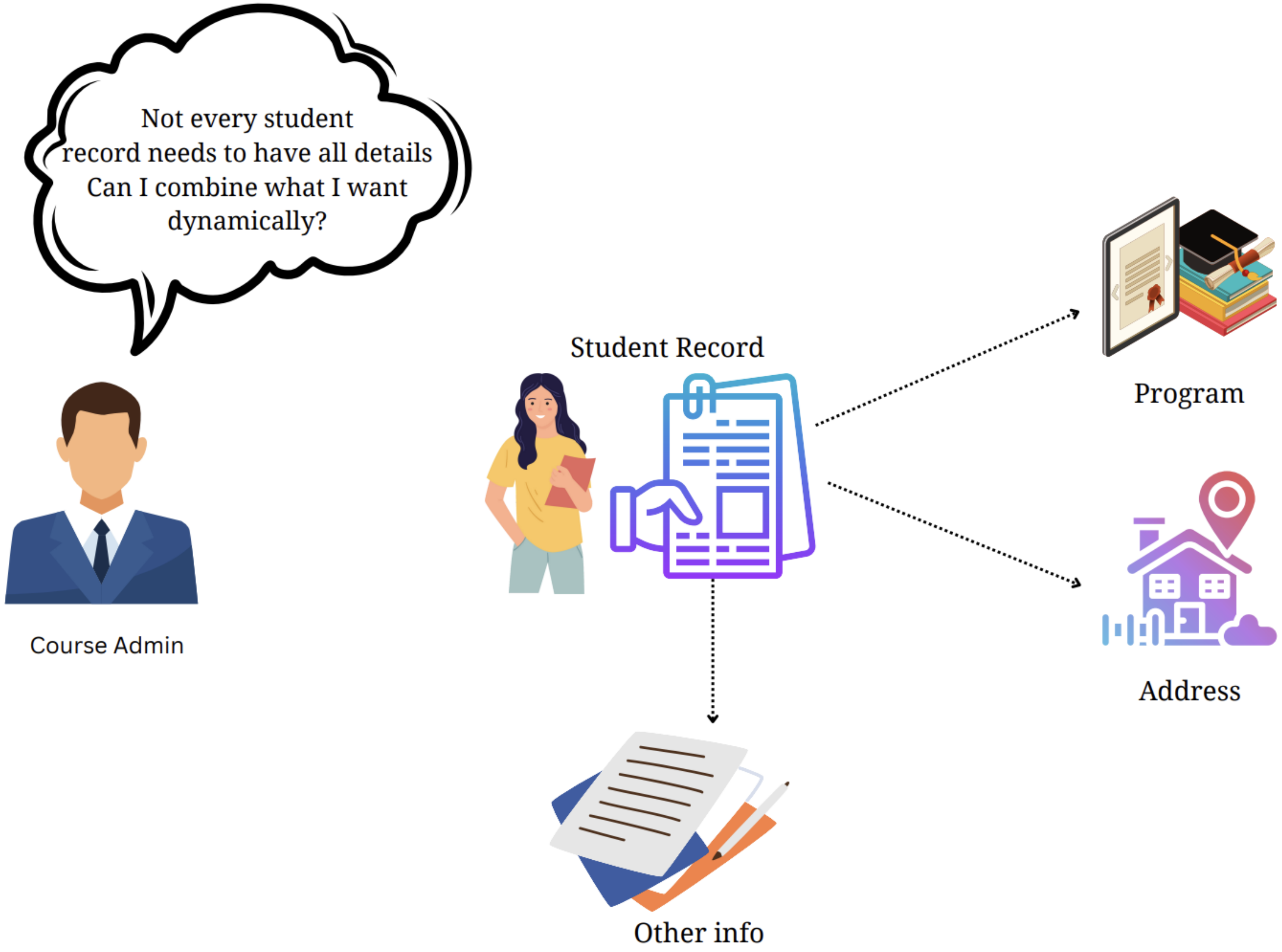
1. **Design Patterns: Elements of Reusable Object-Oriented Software** by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
2. **Head first Design Patterns**, Second Edition, Eric Freeman and Elisabeth Robson

How about building things: Builder Pattern! [Creational]

Meet the Builder Pattern!



Meet the Builder Pattern!


Not every student record needs to have all details Can I combine what I want dynamically?
Course Admin
Student Record
Program
Address
Other info

How to dynamically build the different types of student records?

Meet the Builder Pattern

- What if there is a complex object?
- Can we avoid instantiation of a huge constructor?
- Not every time all constructor parameters are required
- Allows extraction of object construction code to separate object
- Creation of an object is just about assembling other objects step by step
- A very decoupled approach to creation

Builder Pattern: Documentation

Intent

Separate construction of complex object from representation such that same construction process can result in different representations

Also Known As: Builder

Motivation

- Separate object construction from business logic
- Promote readability and understandability
- Three key objects: *Director, Builder, Product*

Example: Builder to build different types of vehicles [Each has engine, tyre, etc]

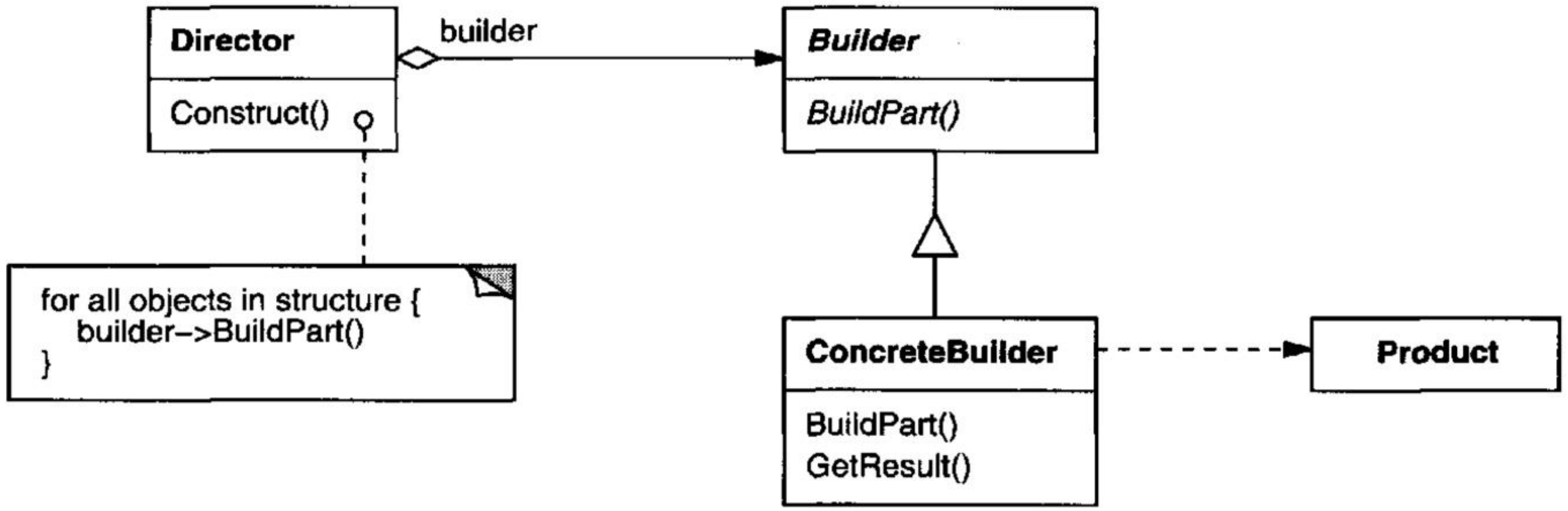
Builder Pattern: Documentation

Applicability

- Algorithm for creating the object must be independent
  - Different parts may make up the object
  - Need not worry about how they are put together
- Construction of different representations of the object needs to be supported

Builder Pattern: Documentation

Structure



```
for all objects in structure {
    builder->BuildPart()
}
```

Image source: Gang of four book

Builder Pattern: Documentation

Participants

Builder (StudentBuilder)

- Defines the interface for creating parts of a product object

ConcreteBuilder (ConcreteStudentBuilder)

- Assembles the parts to create product by implementing builder interface

Director (StudentDirector)

- Constructs an object using the builder interface

Product (Student)

- Complex object under construction
- Includes classes that define the different parts

Builder Pattern: Documentation

Consequences

- Easily vary products internal representation
  - Director gets the abstract interface to build a product
  - All that needs to be done is to define a new kind of builder
- Isolate code for representation and constructions
  - Concrete builder contains code for building a kind of product
  - Directors can reuse builders to build different variants of product
- More control over the construction process
  - Step by step approach under directors control – Focus is on the process
- The overall code complexity increases due to multiple classes
  - Benefits in the long run

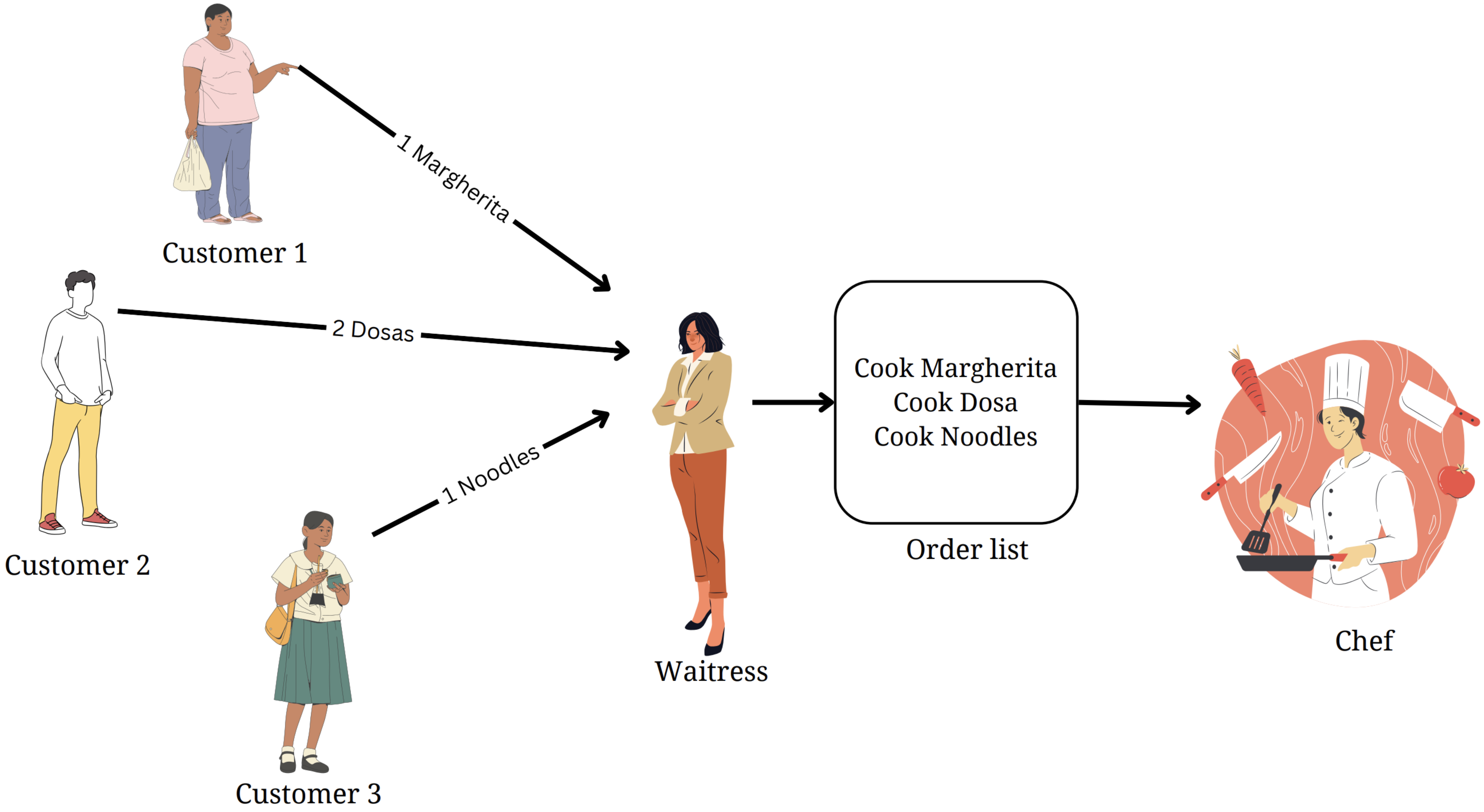
Builder Pattern: Documentation

Implementation

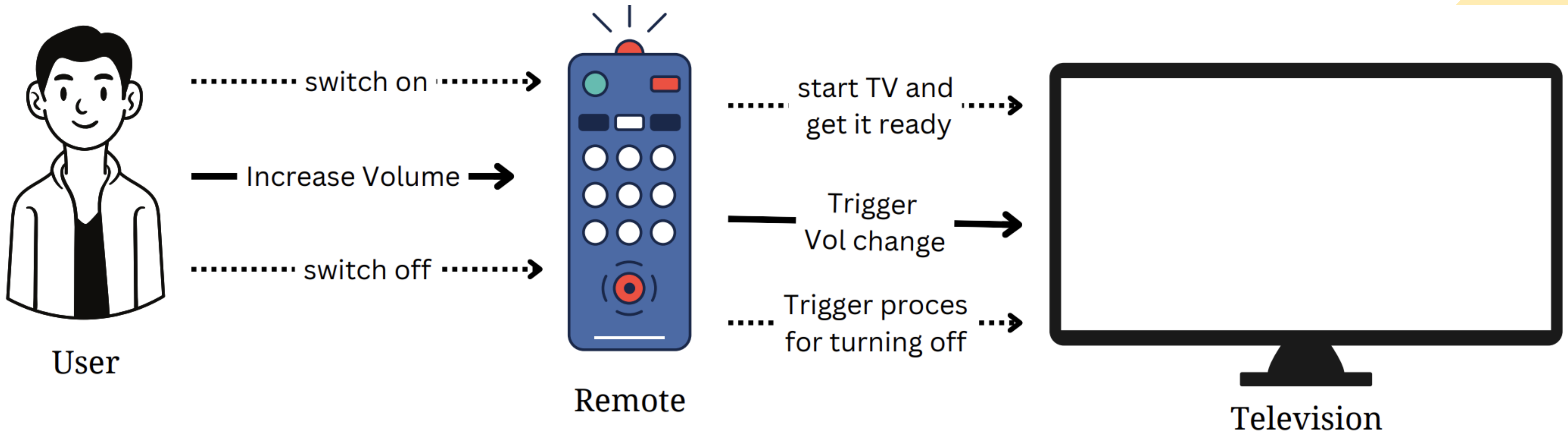
Check the source code given along: StudentRecordBuilder

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?

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

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]

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

Command Pattern: Documentation

Structure

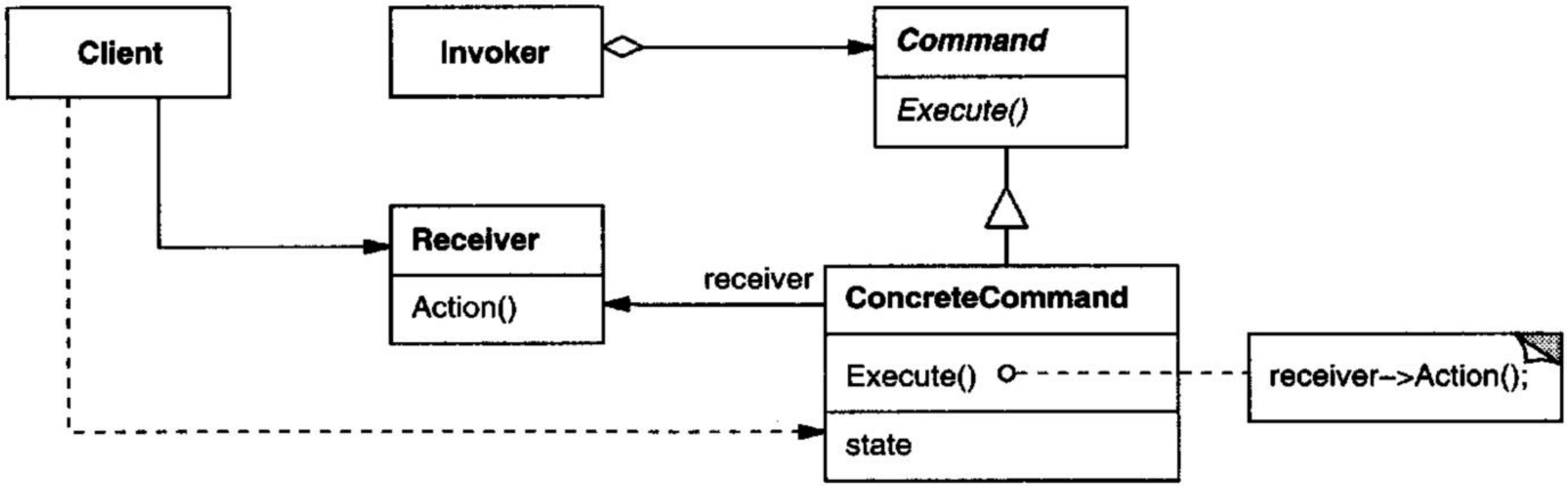


Image source: Gang of four book

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

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

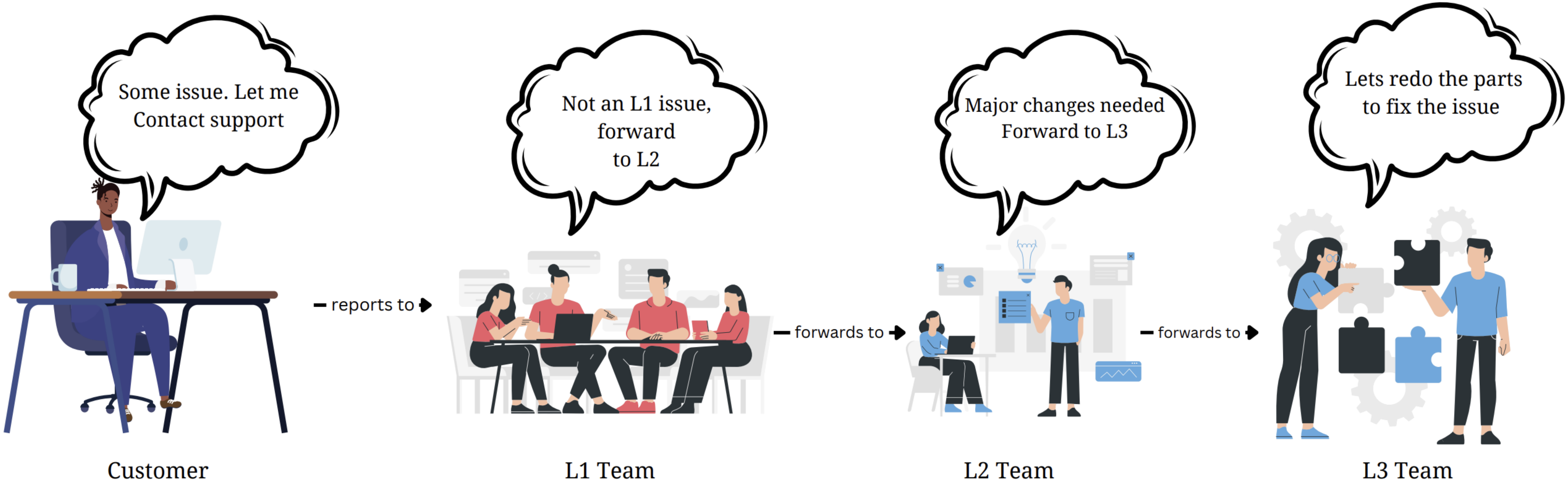
Command Pattern: Documentation

Implementation

Check the source code given along: RemoteControlCommand

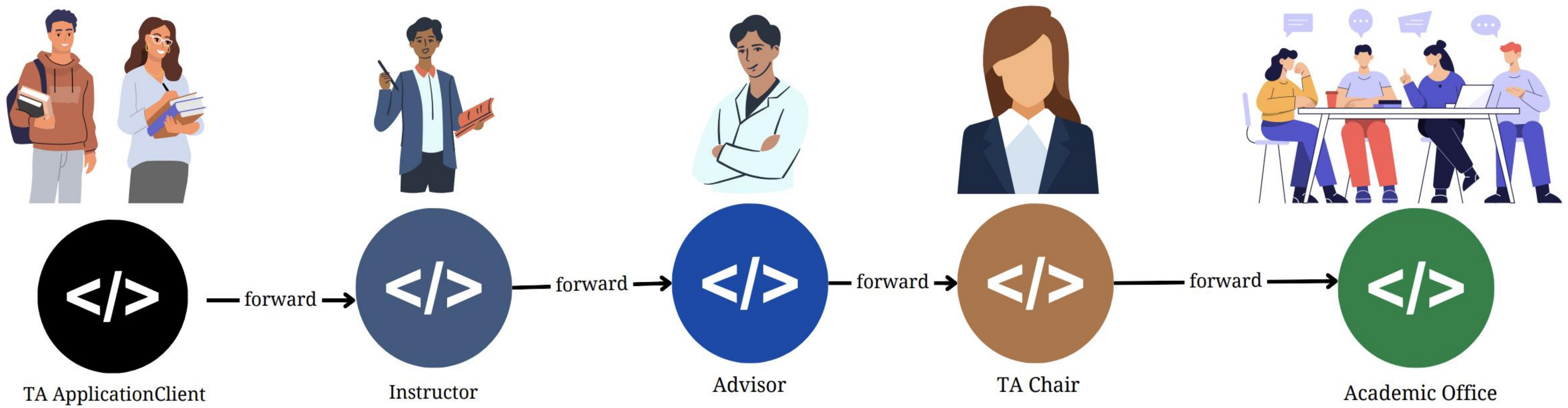
We can pass on responsibilities: Chain of Responsibilities! [Behavioral]

Meet the Chain of Responsibility Pattern!


Some issue. Let me Contact support
Not an L1 issue, forward to L2
Major changes needed Forward to L3
Lets redo the parts to fix the issue
reports to
forwards to
forwards to
Customer
L1 Team
L2 Team
L3 Team

Meet the Chain of Responsibility Pattern - Motivation

TA Application Scenario



How do you implement this ?

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

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

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

CoR Pattern: Documentation

Structure

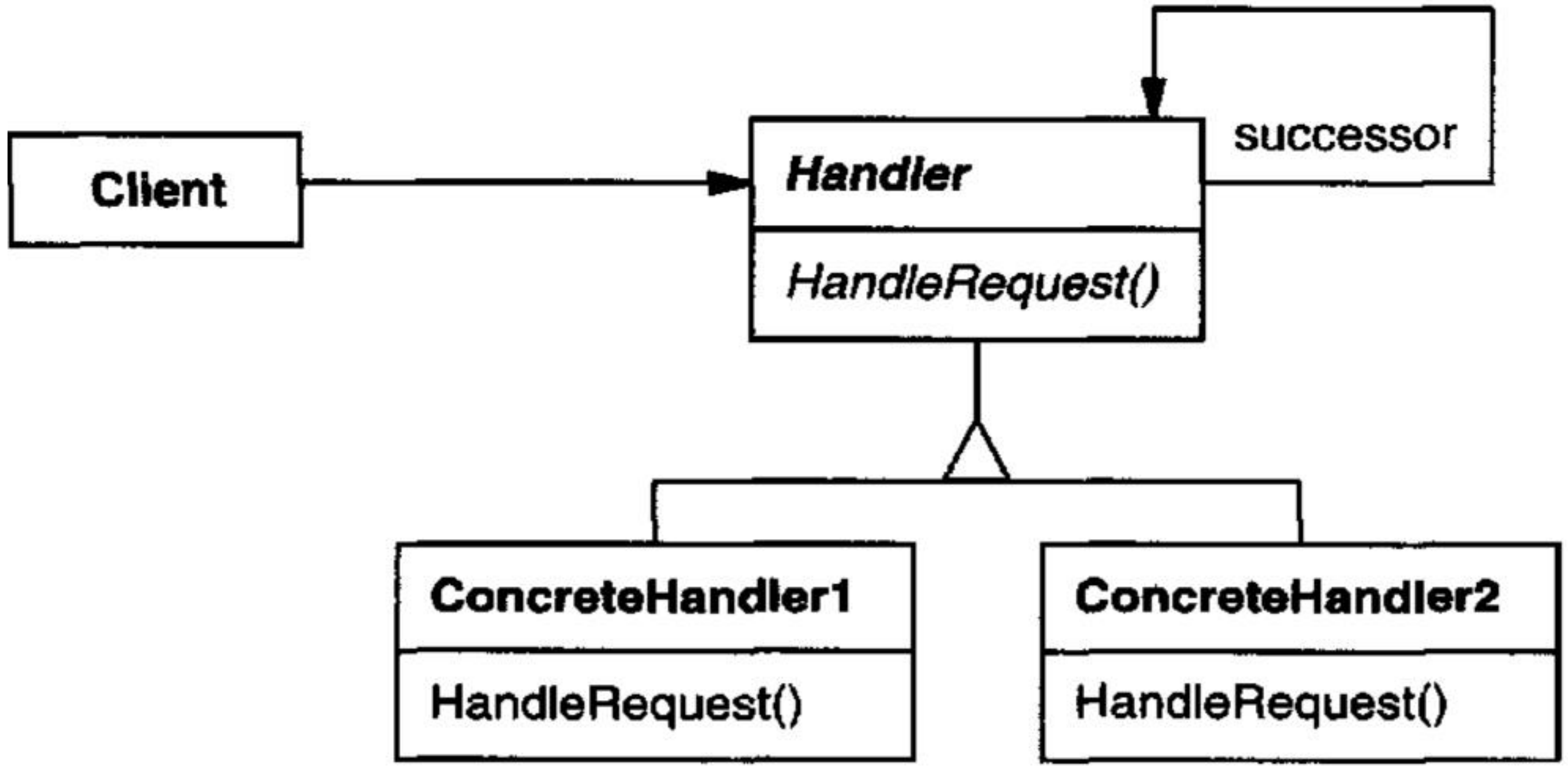


Image source: Gang of four book

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

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

CoR Pattern: Documentation

Implementation

Check the source code given along: TA-ApprovalChainOfResponsibility

Thank You

Course website: karthikv1392.github.io/cs6401_se

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