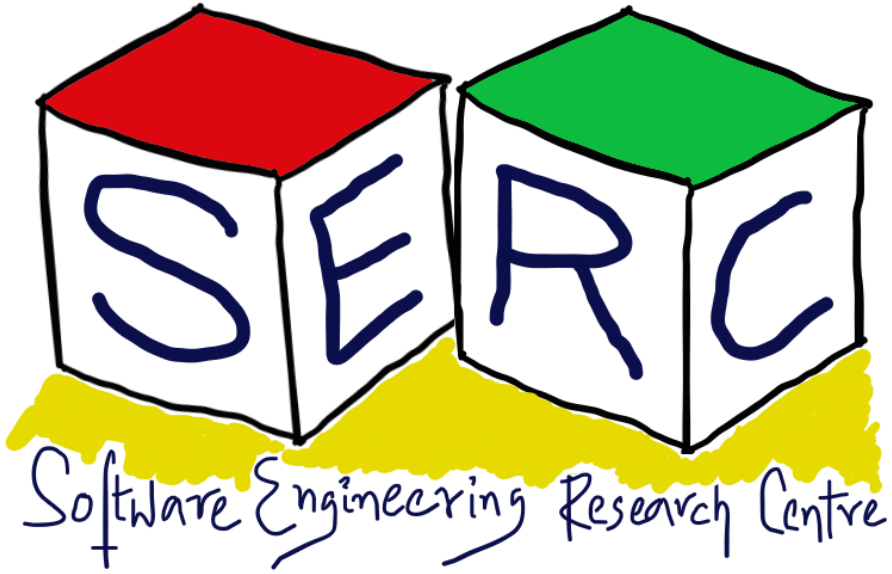
Design Patterns

CS6.401 Software Engineering



Dr. Karthik Vaidhyanathan

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

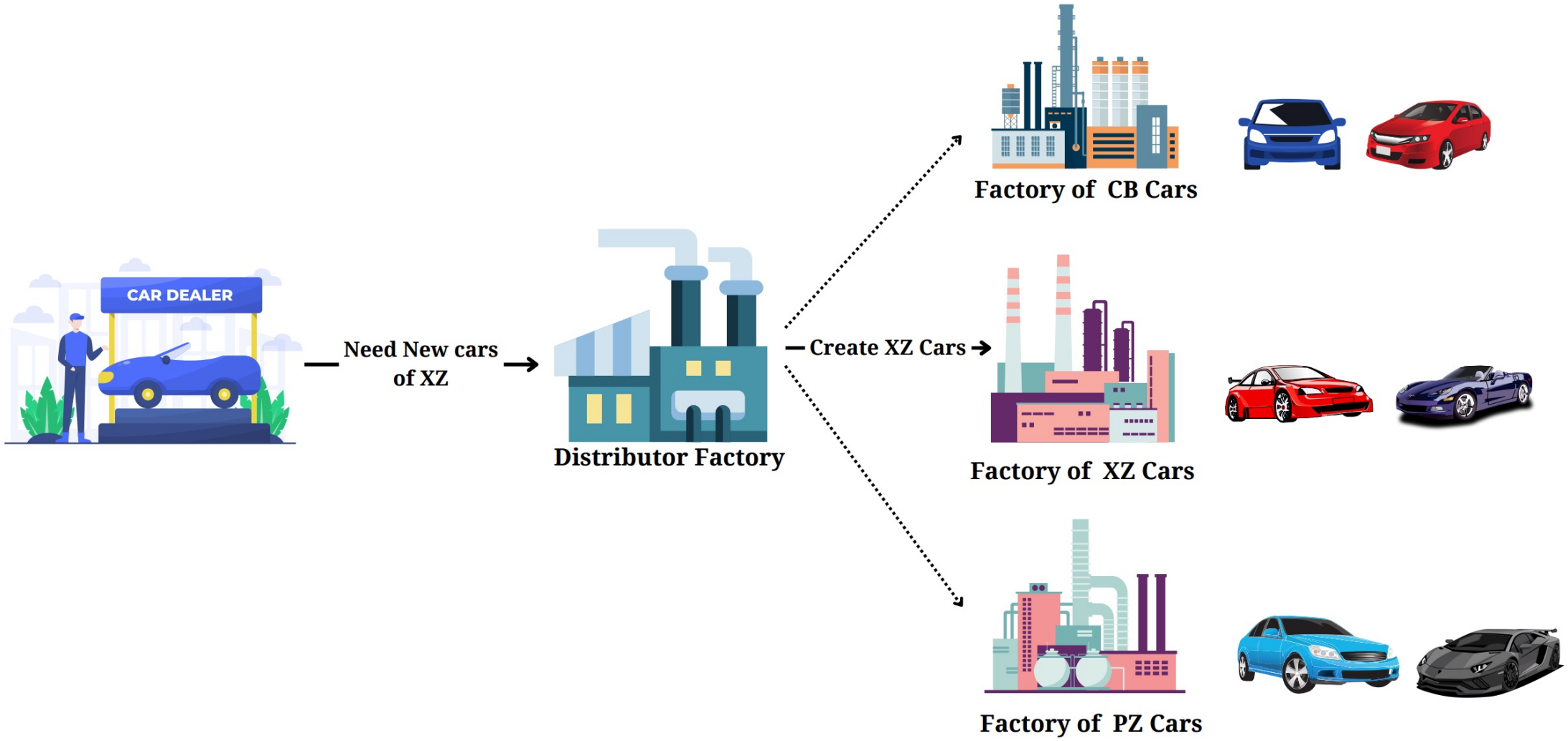
Sources:

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

SERC
Software Engineering Research Centre

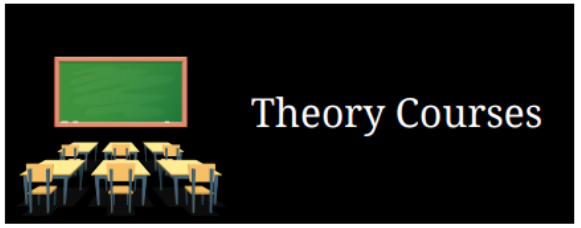
Let’s build a factory to create objects – Factory Pattern! [Creational]

Meet the Factory Pattern!



A distributor may want multiple cars– Just order to the vendor!!

Meet the Factory Pattern: Motivation



Enroll function may be different in each! We may want to add more in future - Elective

SERC
Software Engineering Research Centre

Meet the Factory Pattern

- What if we want to easily add new products (objects of new type)?
- What if you don't want to change too many places when something is added?
- Decoupling clients from knowing actual products (program for interface)
- Encapsulate object creation (encapsulate what varies)

Factory Pattern: Documentation

Intent

Defining an interface for creating object but let subclasses decide which class to be instantiated

Also Known As: Virtual Constructor

Motivation

- Not clear which of the subclasses of the parent class to access
- Encapsulate the functionality required to select a class to method
- Two key objects: *Factory (Creator) and Product*

Factory Pattern: Documentation

Applicability

- A class can’t anticipate the class of objects it must create
- Class wants subclasses to specify the object it creates
- Classes delegate responsibility to one of the several helper classes and which is the delegate needs to be localized

Factory Pattern: Documentation

Structure

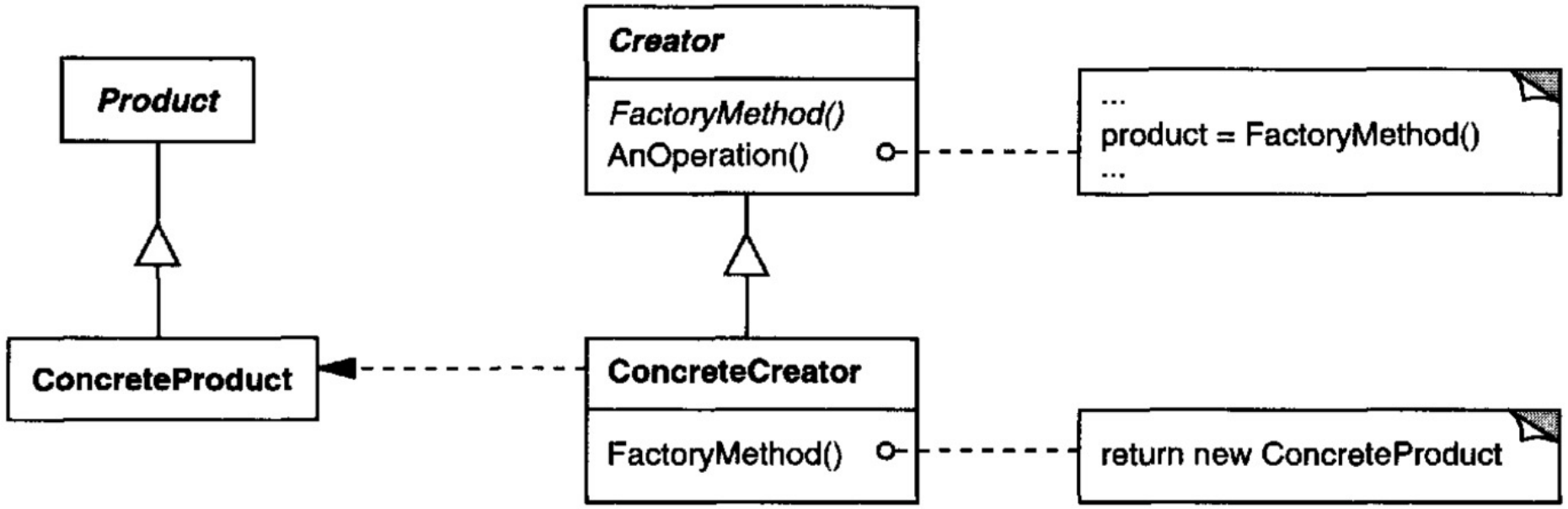


Image source: Gang of four book

Factory Pattern: Documentation

Participants

Product (Systems Interface)

- Defines the interface of objects the factory method creates

Concrete Product (RegularSystemsCourse)

- Implements the product interface

Creator (CourseFactory)

- Declares the factory method which returns object of type product
- Calls factory method to create the product

Concrete Creator (RegularCourseFactory)

- Overrides the factory method to return instance of concrete product

This Photo by Unknown Author is licensed under CC BY

Factory Pattern: Documentation

Consequences

- Eliminates the need to bind application-specific classes into code
  - Code only deals with the product interface
  - Any number of concrete products can be added

- Provides hooks for subclasses
  - Creating objects inside a class is more flexible than direct creation

- Connects parallel hierarchies
  - Class can delegate some of its responsibilities to another class
  - Those can also use the abstract factory

- Too much of subclassing can happen
  - Code can become too complicated
  - Becomes more easier to introduce factory to existing hierarchy

Factory Pattern: Documentation

Implementation

Check the source code given along: CourseFactory

We can always use an adapter: Adapter Pattern! [Structural]

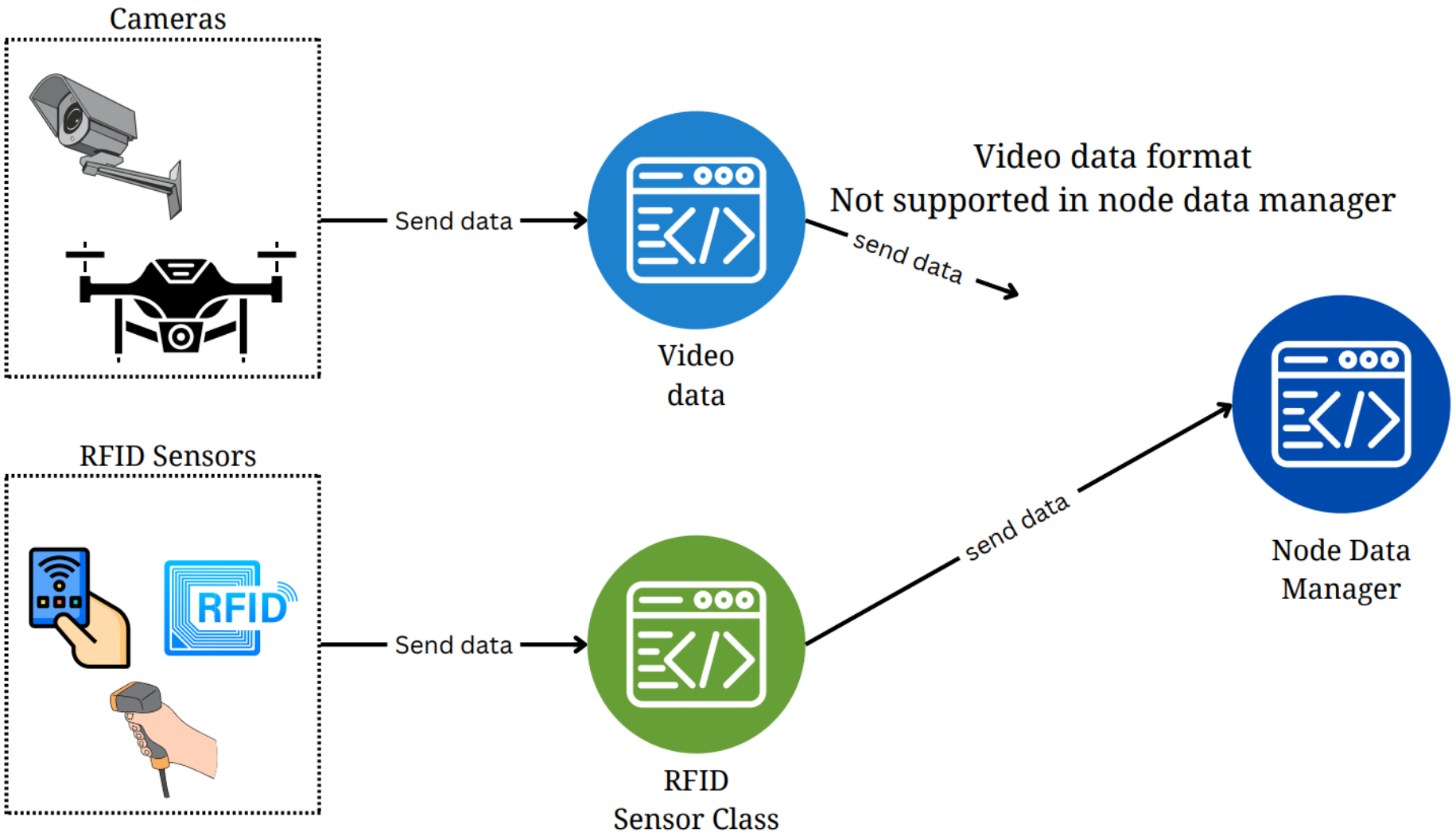
Meet the Adapter Pattern!

Indian

European

Universal adapter

Meet the Adapter Pattern – A Scenario



Why don't we write an adapter that can transform?

Meet the Adapter Pattern

- What if the interfaces are incompatible?
- What if we can have an adapter in between that can transform the new format?
- Adapter wraps the complexity of conversion
- Supports collaboration of different types of object
- Two-way adapter can also be made

This Photo by Unknown Author is licensed under CC BY-SA

Adapter Pattern: Documentation

Intent

Convert the interface of a class into another interface expected by the clients

Also Known As: Wrapper

Motivation

- Not every time there are compatible interfaces
- Promote reusability
- Three key objects: *Client, Target, Adapter*

Example: Adapter to transform data [Think of legacy class that accepts only certain formats]

Adapter Pattern: Documentation

Applicability

- There is an existing class but its interface does not match the one needed
- Creation of reusable class that can work with unforeseen classes
- There are several existing subclasses but impractical to adapt their interface by subclassing everyone
  - Use object adapter [The one we use here] – Uses composition
  - Class adapter relies on multiple inheritance

SERC
Software Engineering Research Centre

Adapter Pattern: Documentation

Structure

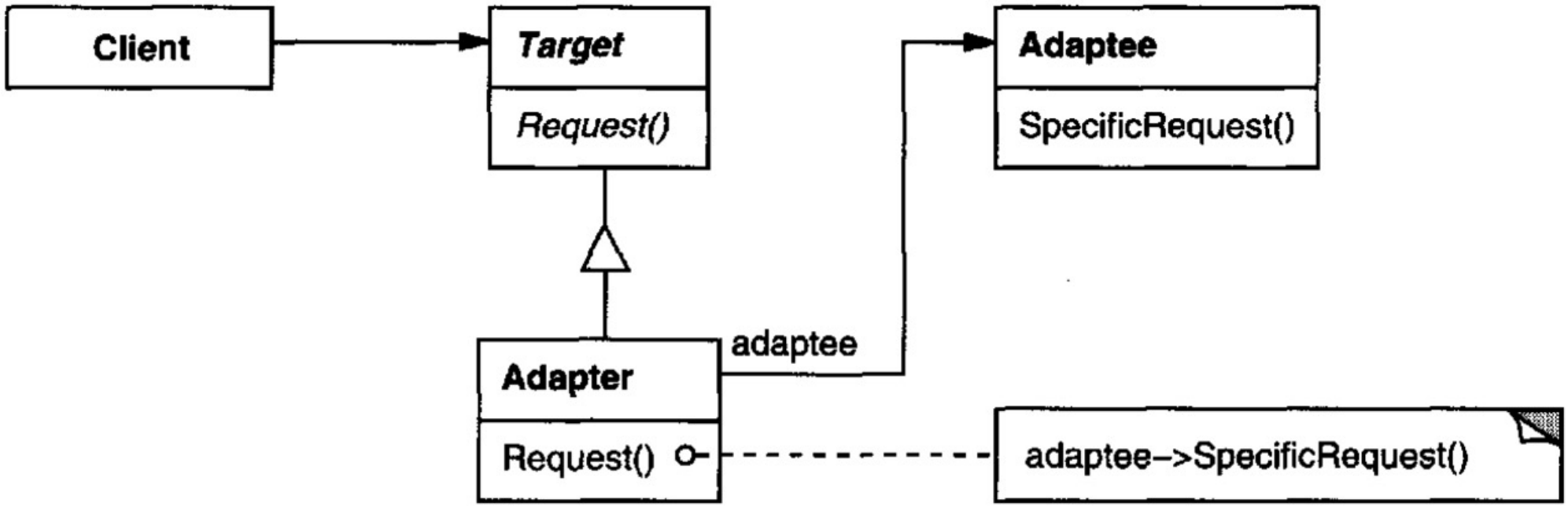


Image source: Gang of four book

Adapter Pattern: Documentation

Participants

Target (NodeData)

- Defines the domain specific interfaces that the client uses

Client (NodeManager)

- Collaborates with objects conforming to their target interfaces

Adaptee (VideoNode)

- Defines an existing interface that needs adapting

Adapter (VideoNodeAdapter)

- Adapts the interface of the Adaptee to the Target interface

This Photo by Unknown Author is licensed under CC BY

Adapter Pattern: Documentation

Consequences

- Single adapter can be used for many adapteees
  - Can implement different functionalities to work with many adaptees
  - New types of adapter can also be easily introduced

- Provides good separation of concerns
  - Keep the logic for conversion in one
  - No need to change at multiple places

- Overall complexity may increase – How much of adaptation is done?
  - Can it be done in a simpler manner on the Adaptee or Target?

Adapter Pattern: Documentation

Implementation

Check the source code given along: IoTAdapter

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

SERC
Software Engineering Research Centre