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:

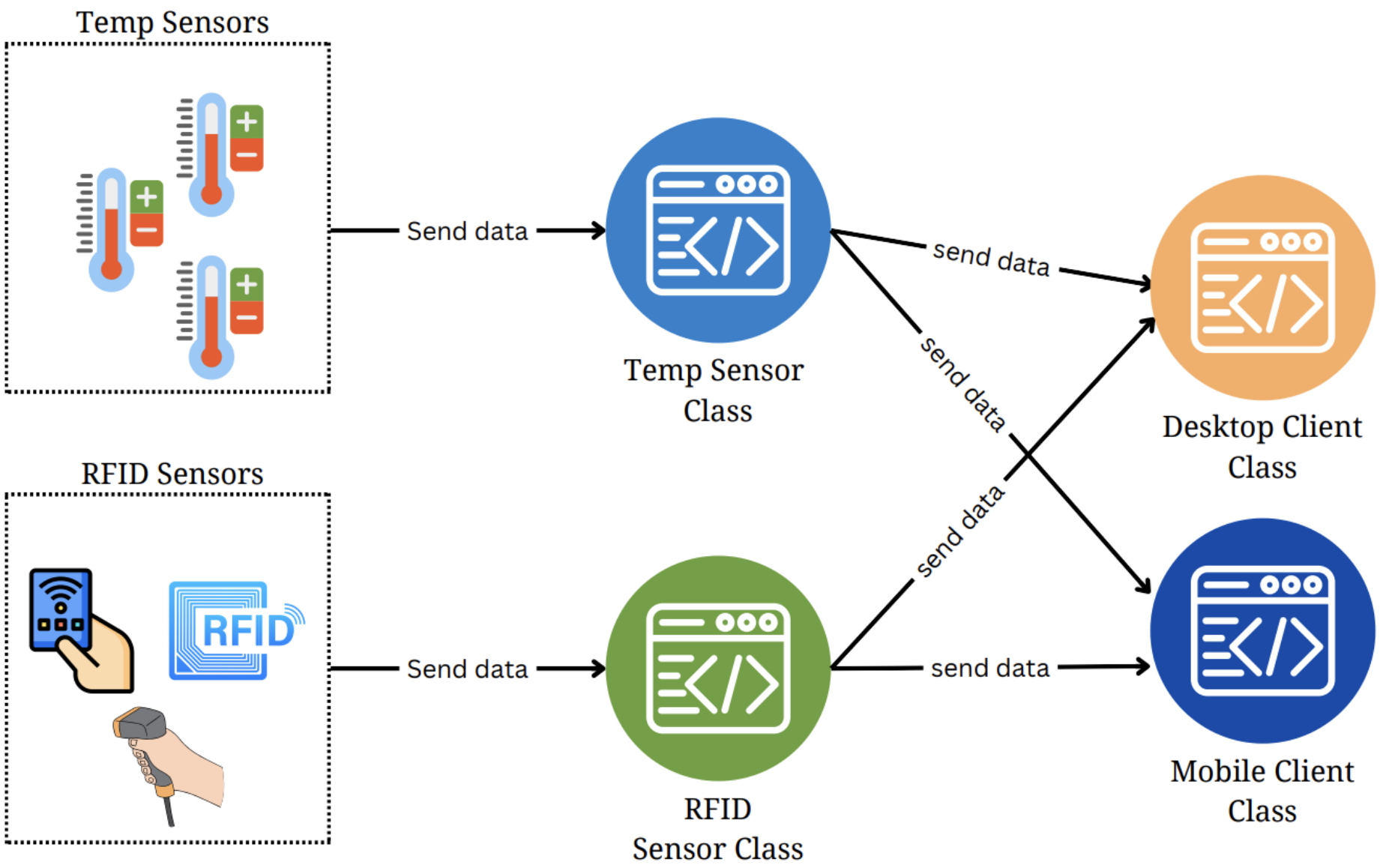
1. **Design Patterns: Elements of Reusable Object-Oriented Software** by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
2. **Head first Design Patterns**, Second Edition, Eric Freeman and Elisabeth Robson

Being an Observer! - The Observer Pattern [Behavioral]

Meet the Observer Pattern!

- Subscriber chooses the (channel) publisher by pressing on subscribe button
- The channel who is posting (Publisher) delivers only to its subscribers
- publisher has to maintain a list of subscribers (channel subscribers)

Meet the Observer Pattern: Motivation



Can we push the data to all clients as soon as it arrives?

Meet the Observer Pattern

- What if we had the sensor data to be publishers?
- What if the clients just become subscribers?
- Every time data comes, all the subscribers are notified
- Publishers and subscribers can be decoupled
- Adding new clients also is just same as adding a new subscriber

Observer Pattern: Documentation

Intent

Defining a one-to-many dependency between objects
Change in object notifies all dependent objects

Also Known As: Dependents, Publish-subscribe

Motivation

- Maintaining consistency between objects
- Reduce tight coupling and increase reusability
- Two key objects: *Subject and Observer*

Example: Presentation components and application data

Observer Pattern: Documentation

Applicability

- When abstraction has two aspects – One dependent on the other and separation promotes reusability
  - Eg: Think of having just one class, Display instead of mobile and web
- When a change in one object requires changing others [Not clear how many!]
- When object should notify others without assuming about the objects [reduce coupling]

Observer Pattern: Documentation

Structure

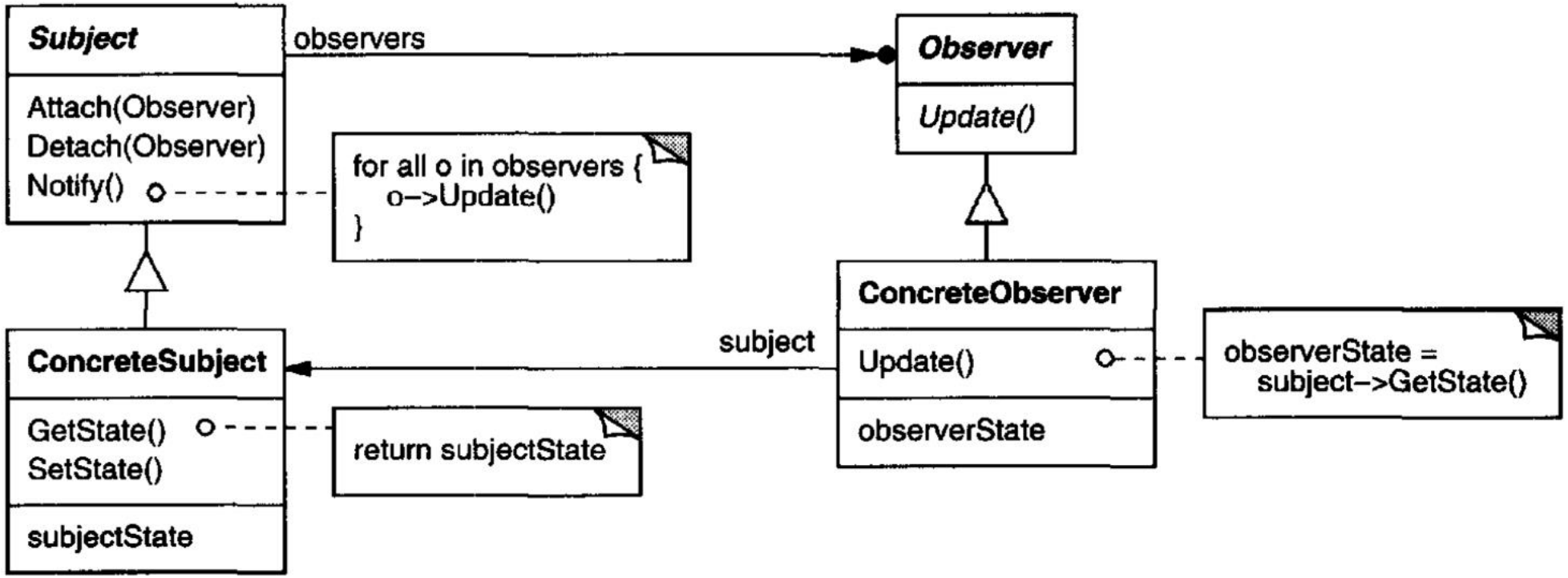


Image source: Gang of four book

Observer Pattern: Documentation

Participants

Subject (ManagerInterface)

- Knows its observers – Many observers per subject
- Provides interface for attaching and detaching observer objects

Observer (DataSubscribers)

- Defines an update interface for objects that should be notified

Concrete Subject (RfidPublisher/Manager)

- The key subject that contains the state information
- Sends a notification to its observers when state change happens

Concrete Observer (MobileSubscriber)

- Maintains reference to concrete subject object
- Implements observer update interface

Observer Pattern: Documentation

Consequences

- Abstract coupling between Subject and Observer
  - Subject doesn't know the concrete class of any observer
  - The coupling is as minimal as possible

- Support for broadcast communication
  - Subject doesn't care about number of observers
  - The notifications are automatically sent as broadcast to all interested

- Unexpected updates
  - Unintended updates on subject may cause cascade of updates on observers
  - Often simple update notification may not provide enough changes on state changes of subject

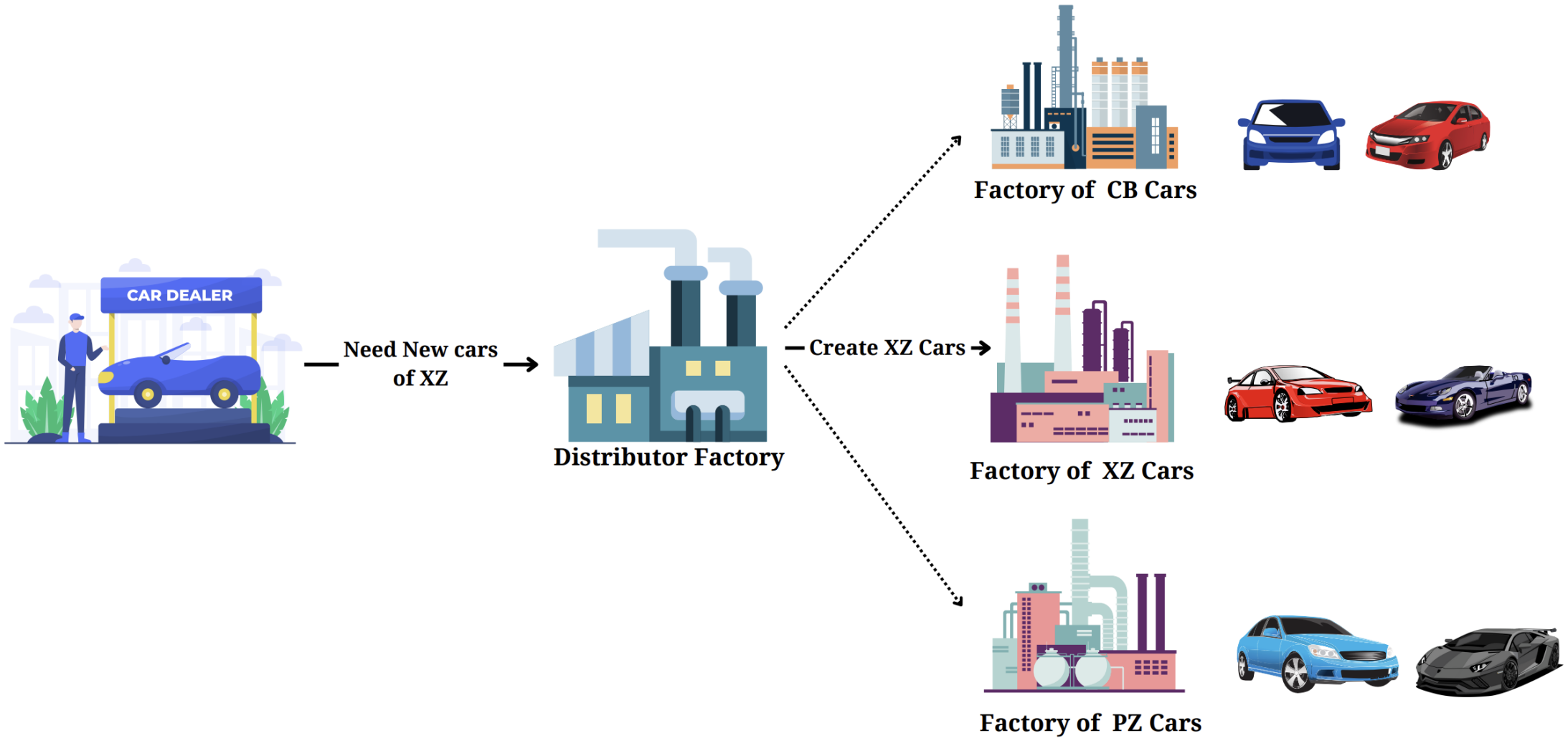
Observer Pattern: Documentation

Implementation

Check the source code given along: IoTObserver

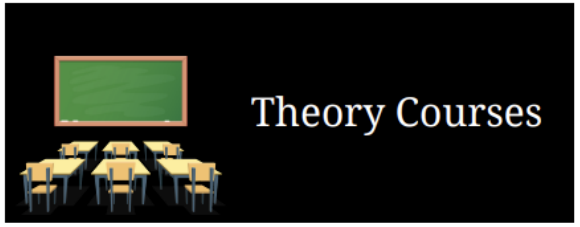
Let’s build a factory to create objects – Factory Pattern! [Creational]

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

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

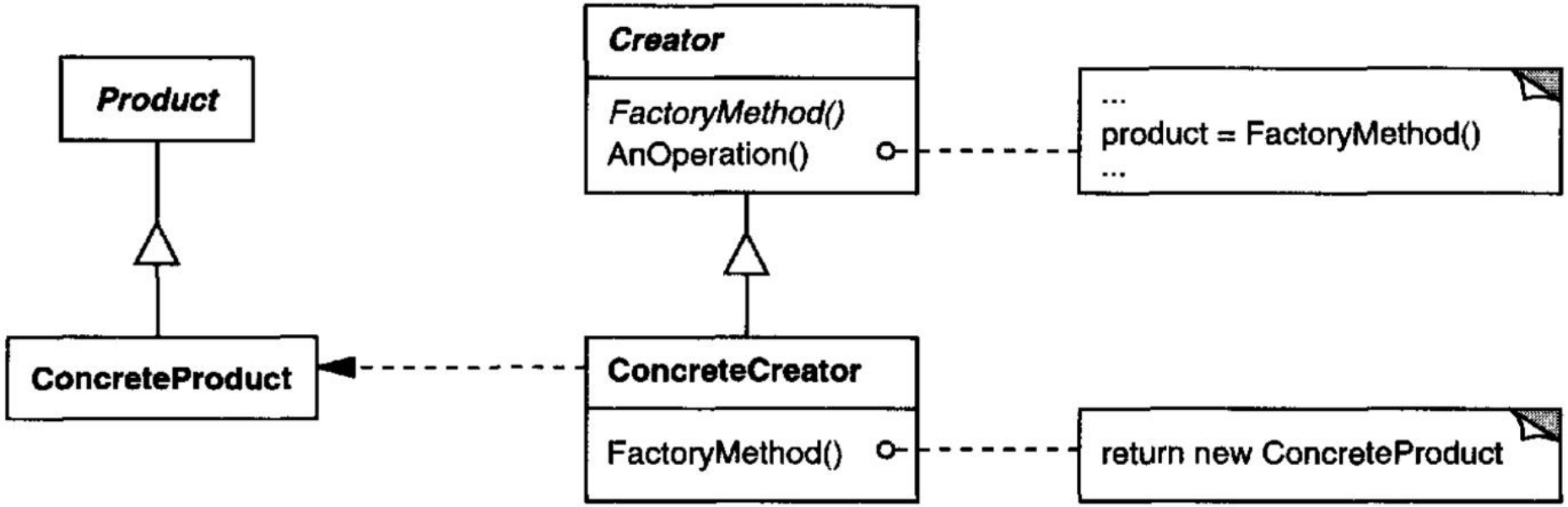


Image source: Gang of four book

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

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

Thank You

Course website: karthikv1392.github.io/cs6401_se

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