Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Object – Oriented Analysis and Design

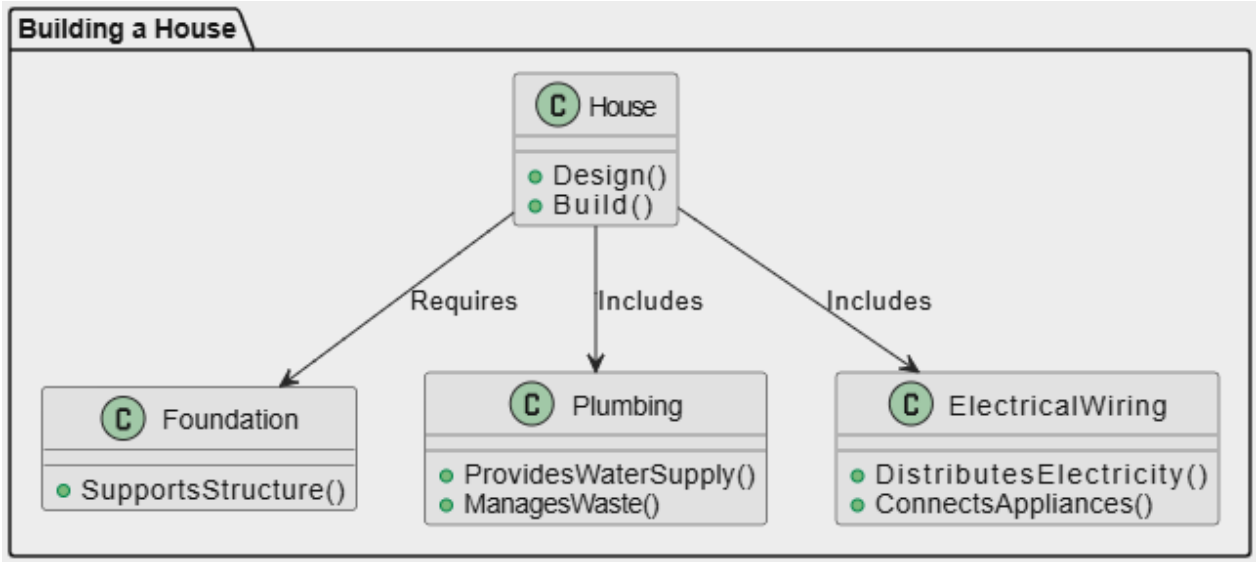
Design Pattern

Instructor: Le Thi Ngoc Hanh, Ph.D

ltnhanh@hcmiu.edu.vn

What are Design Patterns

Design patterns are typical solutions to commonly occurring problems in software design. Design patterns in software development provide established solutions to solve recurring design problems.



Why Design Patterns Matter in Development

- Simplifies Communication: Patterns give developers a shared language to discuss design approaches.
- Improves Code Readability and Maintainability: Patterns provide a standardized way to solve problems.
- Enhances Reusability and Scalability: By reusing solutions, development becomes more efficient.

History and Origins of Design Patterns

1

Christopher Alexander (1977)

"A Pattern Language" introduced the idea of architectural patterns.

2

Gang of Four (1994)

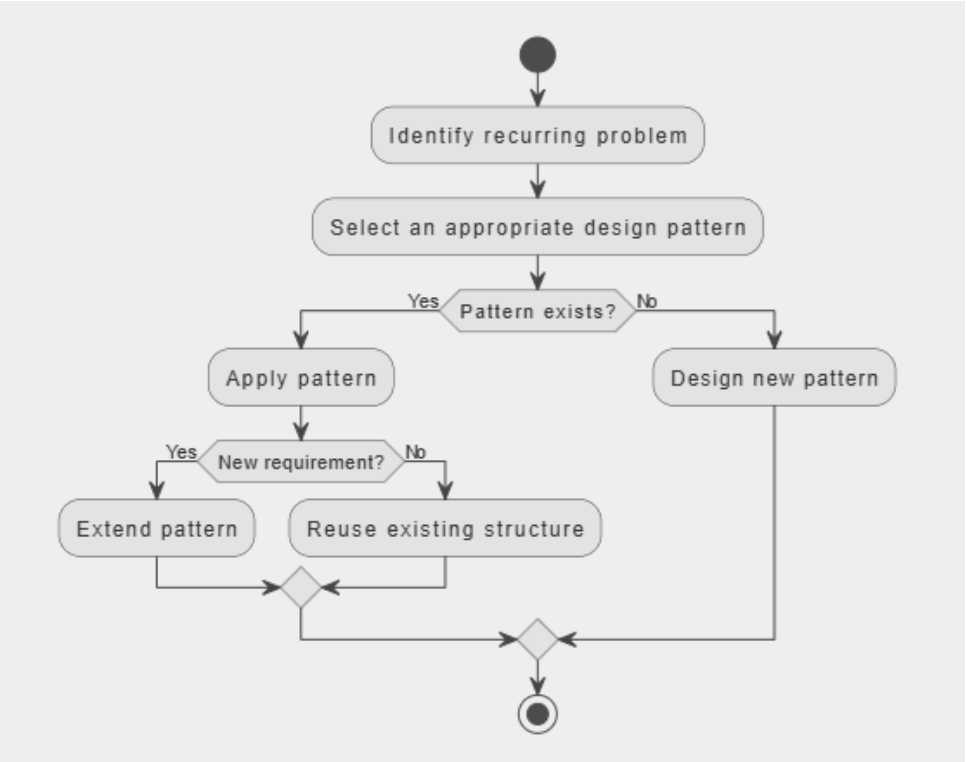
The book "Design Patterns: Elements of Reusable Object-Oriented Software" introduced 23 design patterns.

3

Further Evolution

The field expanded into various domains, including enterprise architecture, UI design, and distributed systems.

How to construct a Design Pattern?

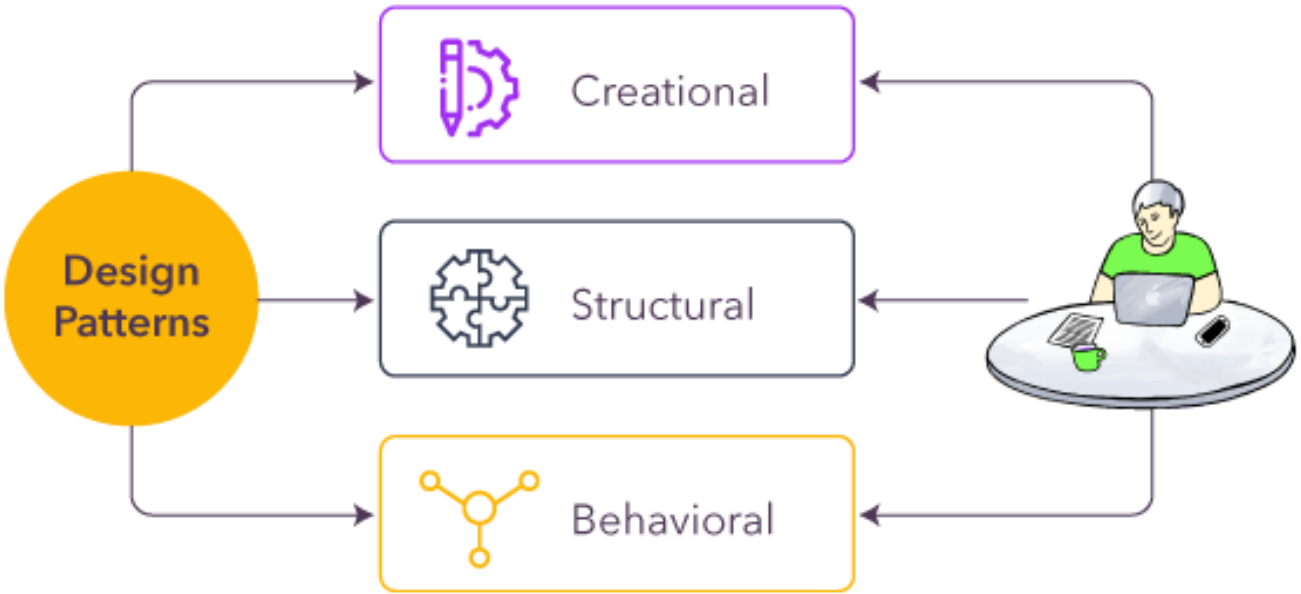


Structure of a Design Pattern

Four essential components of a design pattern:

- **Name**: A concise identifier for easy reference (e.g., Singleton, Observer).
- **Problem**: The specific issue or recurring situation the pattern addresses.
- **Solution**: The approach or structure that resolves the problem.
- **Consequences**: The pros and cons of using this pattern.

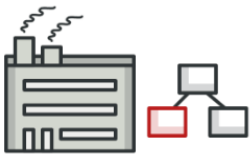
Patterns



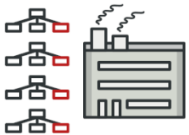
Source: https://www.jobsity.com/blog/design-pattern-software

Creational Patterns

Define Creational patterns as solutions focused on *efficient object creation*.

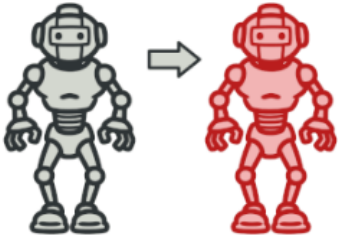


Factory Method

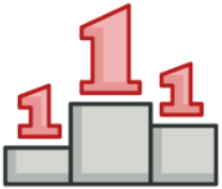


Abstract Factory

Builder



Prototype



Singleton

Image credit: https://refactoring.guru/

Factory Method Pattern

- Factory Method: a *creational design pattern* that provides a way to create objects without specifying the exact class of the object being created.

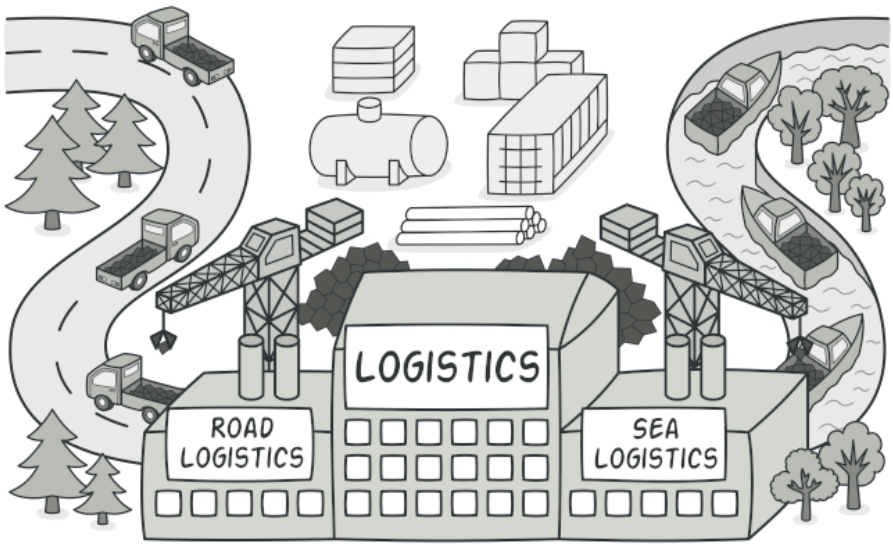


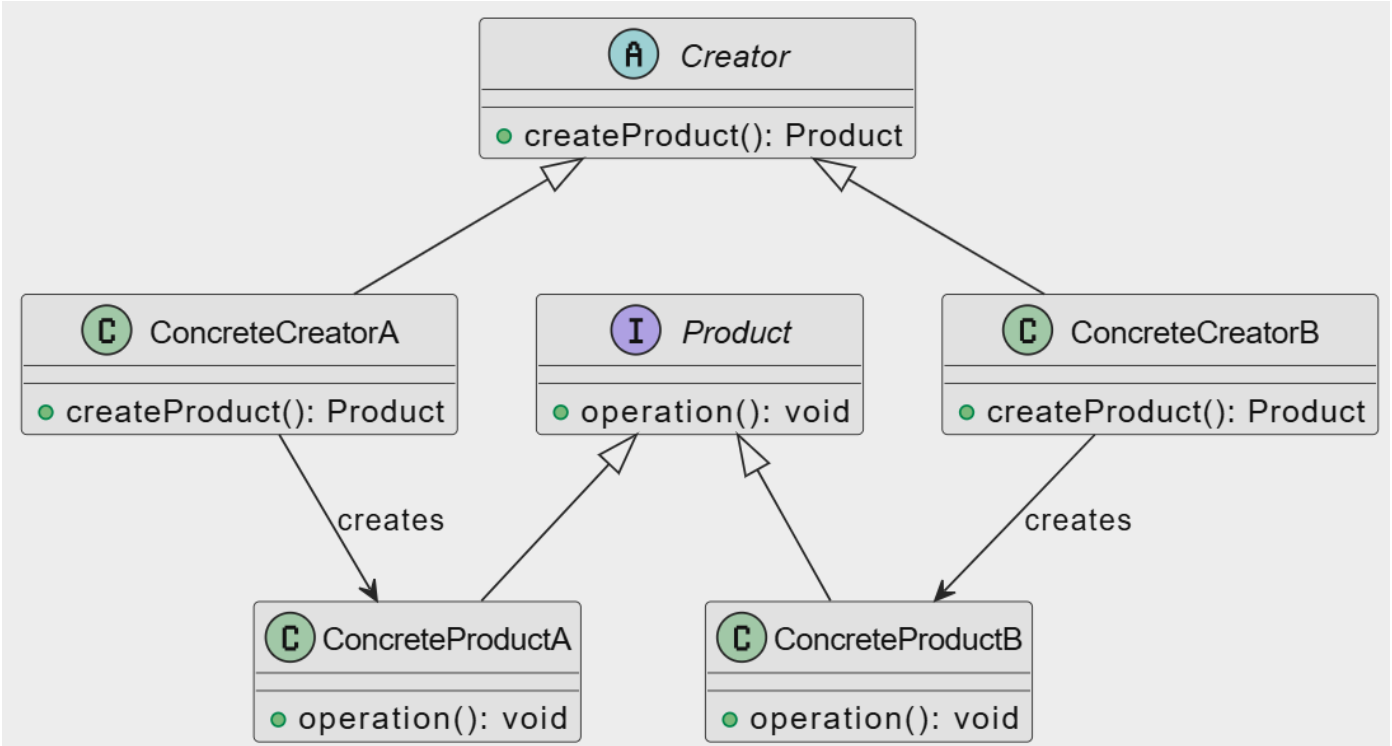
Image credit: https://refactoring.guru/

Factory Method Pattern

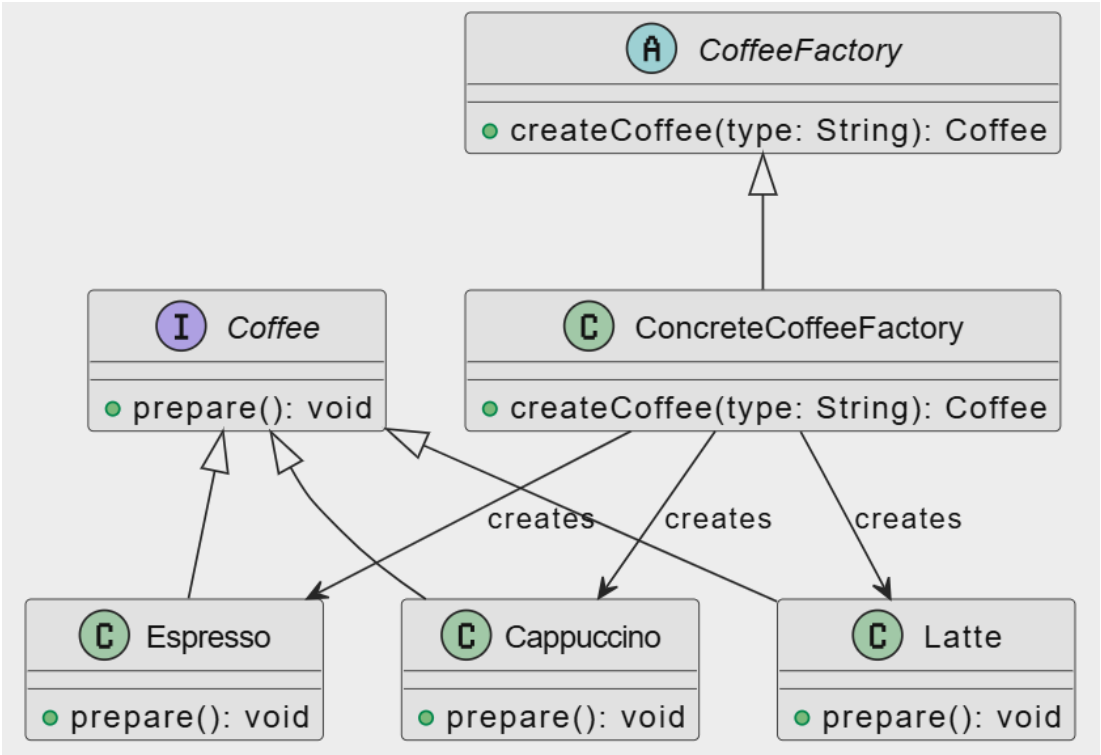
Components:

- Product Interface: Defines the type of object to be created.
- Concrete Product Classes: Implement the Product interface for specific types.
- Creator (Factory) Class: Contains the factory method that creates and returns objects

Factory Pattern

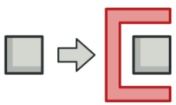


Factory Pattern - Coffee Shop

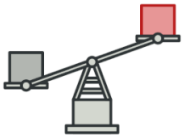


Structural Patterns

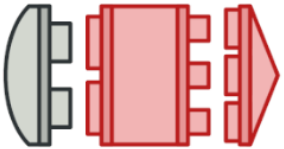
An approaches that help *assemble objects and classes into larger structures*.



Proxy

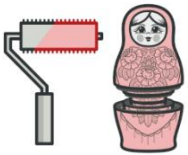


Flyweight



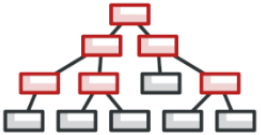
Adapter

Facade



Decorator

Bridge



Composite

Image credit: https://refactoring.guru/

Adapter Pattern

Adapter Pattern acts as a bridge between two incompatible interfaces, enabling them to collaborate without modifying their source code.

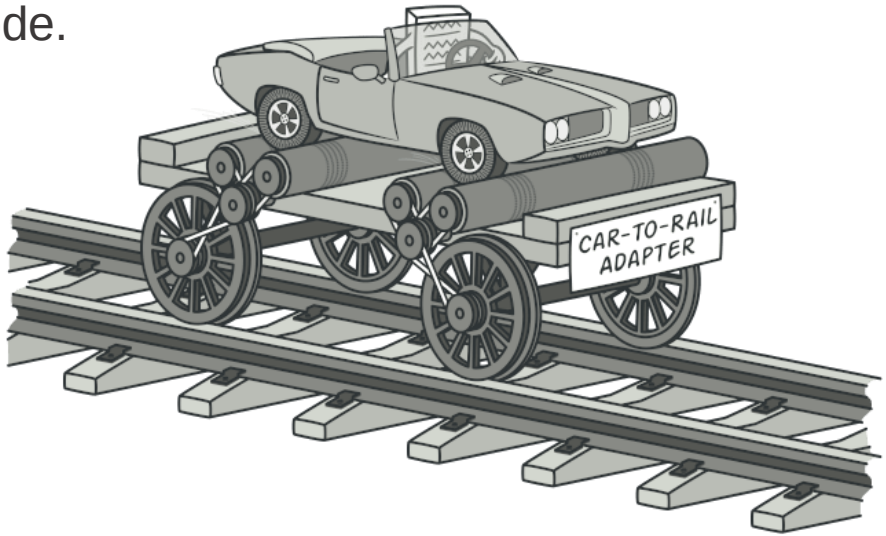


Image credit: https://refactoring.guru/

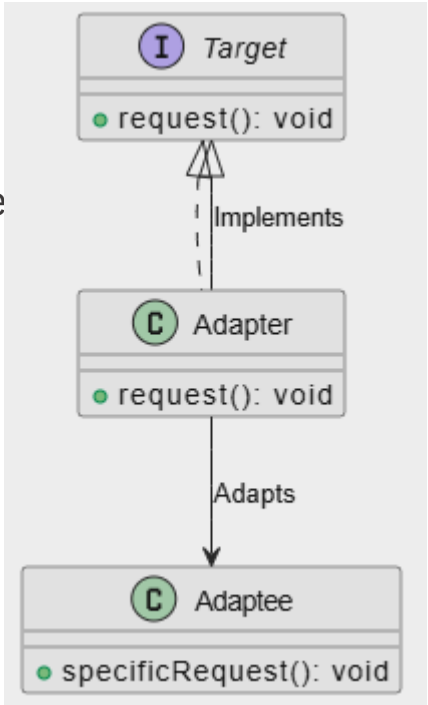
Adapter Pattern

Components:

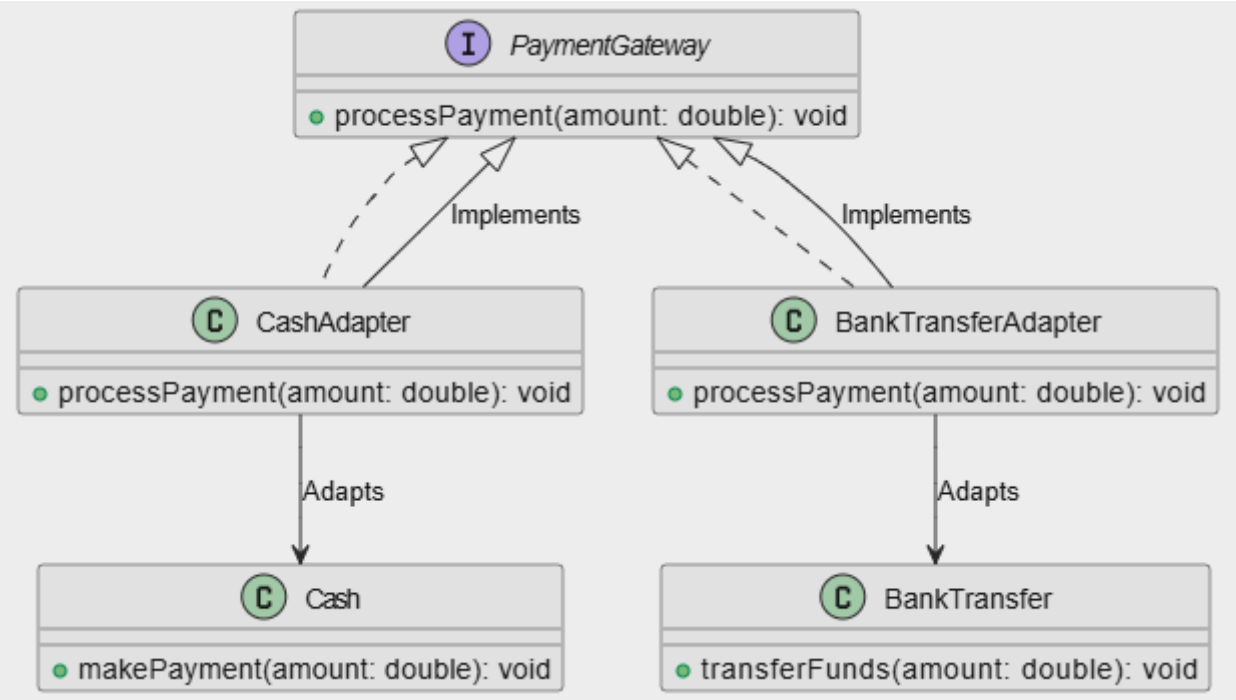
Target Interface: Defines the expected interface by the client (the system needing the adaptation).

Adaptee: The existing interface or class that requires adaptation.

Adapter: A class that implements the Target interface and calls the Adaptee's methods, translating requests where necessary.



Adapter Pattern



Why Adapter Pattern?

- Incompatibility: When two systems/components (interfaces) are incompatible and need a bridge to communicate.
- Reusability: To avoid rewriting or modifying existing classes to fit new requirements.
- Flexibility: To allow systems with incompatible interfaces to work together dynamically.

Behavioral Patterns

Behavioral patterns as methods to *manage communication and responsibility among objects*.

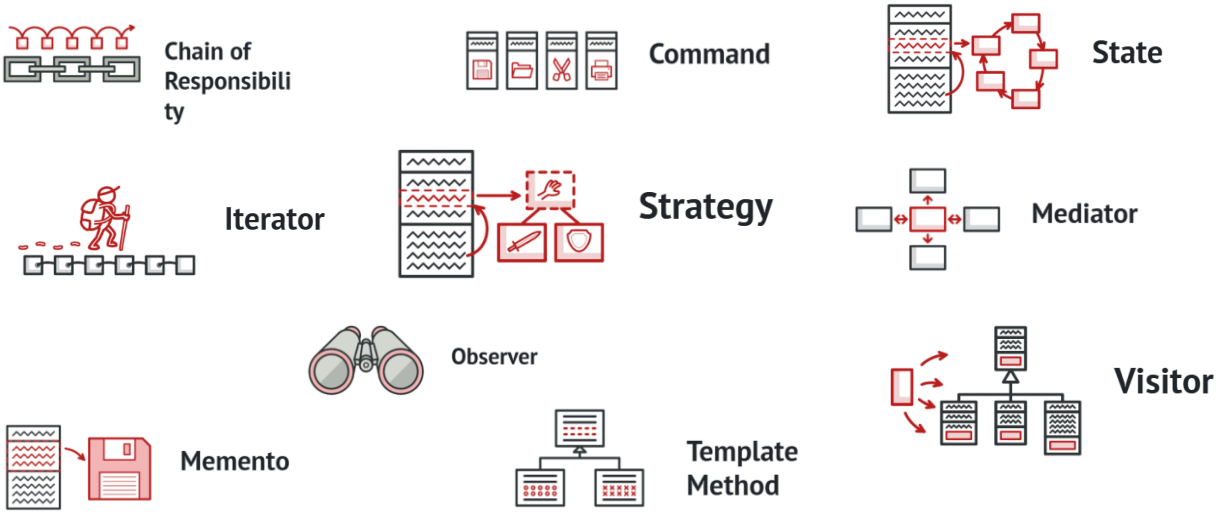


Image credit: https://refactoring.guru/

Observer Pattern

- The Observer Pattern is a behavioral design pattern where an object (the Subject) maintains a list of dependents (the Observers) and notifies them of any state changes.
- Promotes a one-to-many relationship between objects.

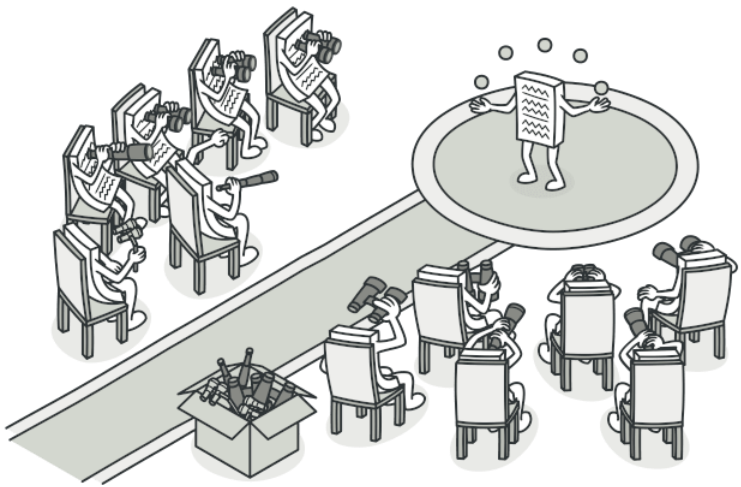
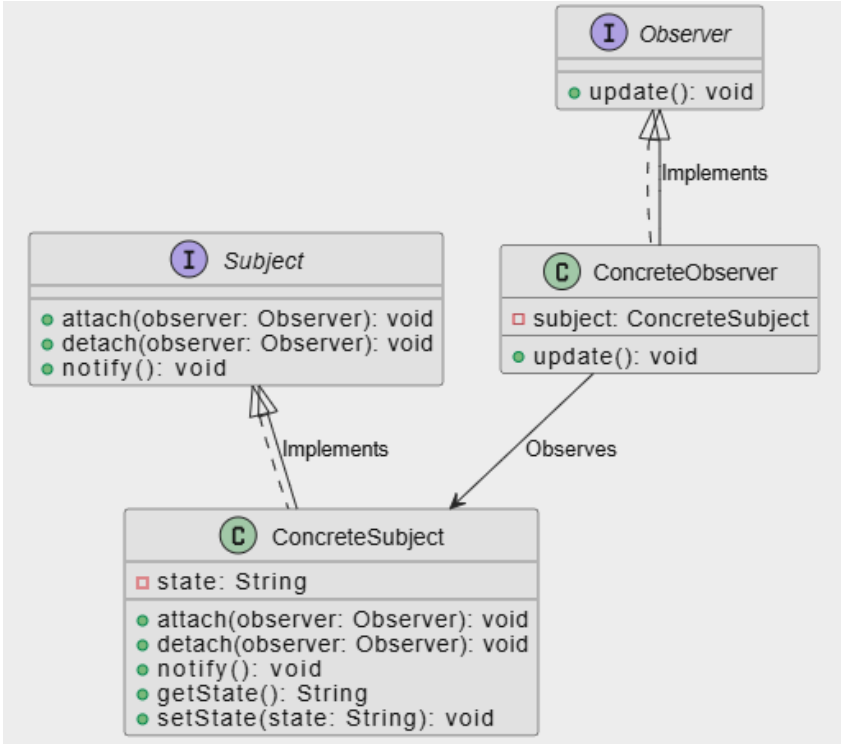


Image credit: https://refactoring.guru/

Observer Pattern

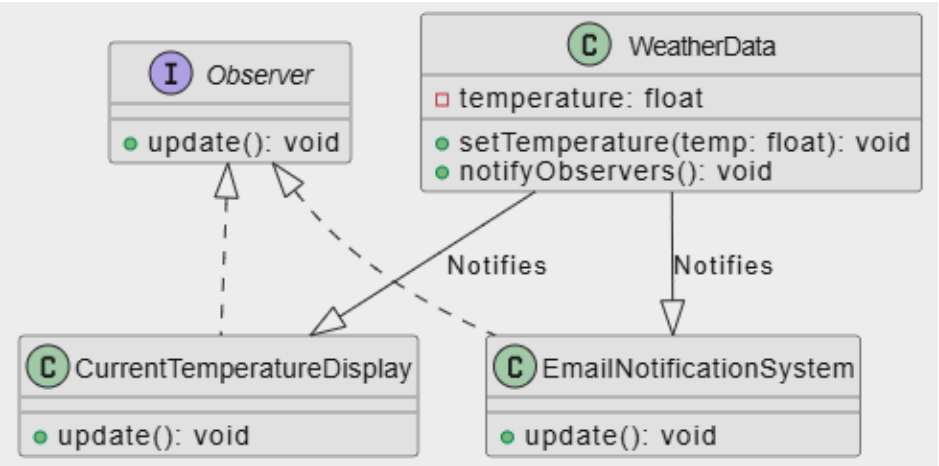
Components:

- Subject: Maintains a list of Observers and notifies them of state changes.
- Observer: Defines an interface for objects that should be notified of changes.
- ConcreteSubject: Implements the Subject interface and stores state.
- ConcreteObserver: Implements the Observer interface and updates its state based on notifications.



Observer Pattern

Example: a **Weather Station** that collects weather data, such as temperature. You want to display this information to different viewers, like a **Current Temperature Display** and an **Email Notification System**.



Discussion

Pitfalls

Without

Patterns?