0. Introduction to Design Patterns

Created	@September 20, 2022 4:36 PM
Progress	In Progress

Software pattern and pattern history `1.1.` `1.2.`

Design pattern categories `1.3.`

- `Creational`, `Structural`, `Behavioral`

Benefits of patterns `1.4.`

- Reusable design
- Communication language

Benefits of patterns `1.5.`

- `Architectural pattern`, `Design pattern`, `Coding pattern`

학습 TODO list

□ design pattern → more productive, flexible, reuable 예

□ collection of objects

□ 1.1.1. 3번 uml 해석

□ 객체 collection

□ run-time 정의

□ 1.3.1. static relationships, patterns of communication

□ 1.3.1. GoF pattern

□ entity

□ composition

□ OO Basics

□ micro-architecture

□ low-level pattern

1.1. What is a pattern?

> ***pattern****: a solution to a problem in a context*

→ applicable to `software development`

1.1.1. Three part rules of Design Pattern

1. `context`: the ***recurring*** situation in which the pattern applies

   ex. ***collection of objects***

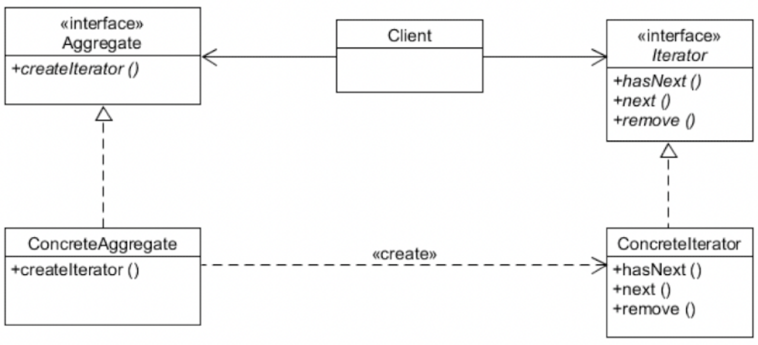
2. `problem`: the ***goal*** you are trying to achieve in this context and any ***constraints*** that occur in the context

   ex. step through the objects ***without exposing the collection's implementation***

3. `solution`: what you after (a ***general design*** that anyone can apply which resolves the goal and the set of constraints)

   - structure with components and relationships
   - ***run-time mechanism***

   ex. ***encapsulate the iteration*** into a separate class



1.2. Software pattern history

- **1977**: *The architect Christopher Alexander, A Pattern Language: Towns, Buildings, Construction*
- **1987**: *Kent Beck and Ward Cunningham, "A Laboratory For Teaching Object-Oriented Thinking", OOPSLA, 1987.*
  - adopted Alexander's pattern idea for Smalltalk GUI design
- **1991**: *Erich Gamma, Ph. D. thesis*
- **1995**: *Gamma, Helm, Johnson, Vlissides (Gang of Four), Design Patterns: Elements of Reusable Object-Oriented Software*
- **1994-**: Pattern Languages of Programs (PLoP) Conferences and books

---

1.3. Design pattern categories

1.3.1. Category of GOF Patterns

			Purpose	
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter	Interpreter Template
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	CoR Command Iterator Mediator Memento Observer State Strategy Visitor

- `Creational`: Address problems of creating an object in a flexible way
  - ***separate creation from operation/use***
- `Structural`: Address problems of ***Object Oriented constructs*** like inheritance to organize classes and objects
- `Behavioral`: Address problems of assigning ***responsibilities*** to classes
  - suggest both ***static relationships*** and ***patterns of communication***

1.4. Benefits of patterns

> *Why do we use patterns?*

> *"Designing object-oriented software is hard, **designing reusable object-oriented software is even harder**"* - Erich Gamma

- ***Experienced designers reuse solutions*** which were proved to work in the past.
- Well-structured object-oriented systems have ***recurring patterns of classes and objects***.
- Knowledge of patterns allow a designer to be ***more productive*** and the resulting designs to be ***more flexible and reusable***.
- Facilitate communication among developers by providing a ***common language***.
- Someone has already solved your problems.

1.4.1. Key Features of Design Patterns

- `Pattern name` : a concise, meaningful name for a pattern improves ***communication*** among developers
- `Intent` : the ***purpose*** of the pattern
- `Problem` : the problem that the pattern is trying to solve
- `Solution` : how the pattern provides a solution to the problem in the context where it shows up
    - emphasizes their ***relationships, responsibilities and collaborations***; rather an abstract description
- `Participants and collaborators` : the ***entities*** involved in the pattern
- `Consequences` : the pros and cons of using the pattern
    - includes impacts on ***reusability, portability, extensibility***
- `Implementation` : how the pattern can be implemented
    - implementations are just concrete manifestations of the pattern and ***should not be considered as the pattern itself***
- `Generic structure` : a standard diagram showing a typical structure for the pattern

---

1.5. Levels of patterns

1.5.1. Hierarchy of Pattern Knowledge

1️⃣ **Design Pattern**
ex. Strategy Pattern: defines a family of algorithms, encapsulates each one, and makes them interchangeable → lets the algorithm vary independently from clients using it

2️⃣ **OO Principles**

- ***encapsulate*** what varies
- favor ***composition*** over inheritance
- program to ***interface***, not implementations

3️⃣ **OO Basics**

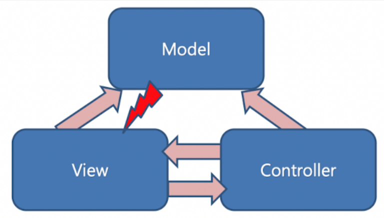
- `Abstraction`
- `Encapsulation`
- `Polymorphism`
- `Inheritance`

1.5.2. Pattern-Oriented Software Architecture (POSA)

1️⃣ `Architectural pattern`

- ***fundamental structural*** organization or schema
- provides ***predefined subsystems***, specifies their ***responsibilities***, and includes ***rules and guidelines for organizing relationships*** between them
- ***affects the overall*** skeletal structure and organization of a software

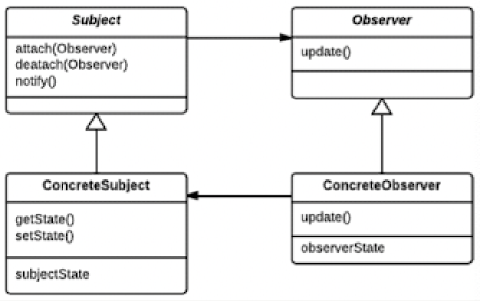
ex. MVC



2️⃣ `Design pattern`

- refine subsystems or components, or relationships between them
- describes commonly recurring structure of components that solves a general design problem within a particular context
- ***does not influence overall*** system structure, but instead define ***micro-architectures*** of subsystems and components

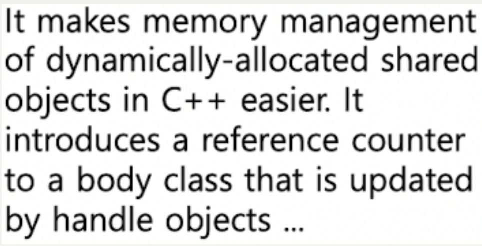
ex. Observer Pattern



3️⃣ `Coding pattern` **(or programming idiom)**

- ***low-level pattern*** specific to a programming language
- describes ***how to implement*** particular aspects of components or the relationships between them using the features of the given language

ex. Counter Pointer



It makes memory management of dynamically-allocated shared objects in C++ easier. It introduces a reference counter to a body class that is updated by handle objects ...

1.6. Quiz

❓ 다음 중 잘못된 설명은?

- □ 설계 작업에서 설계패턴의 이름을 통해 보다 명확하게 의사전달을 할 수 있다.
- □ 설계패턴은 설계 시 자주 반복되는 문제에 대한 해결책을 담고 있다.
- ☑ ~~아키텍처 패턴은 주로 컴포넌트 내부의 설계에 사용되며, 설계패턴은 시스템의 전체 구조를 결정하는데 사용된다.~~

→ ***아키텍처 패턴: 시스템의 전체 구조 결정***

→ ***설계패턴: 주로 컴포넌트 내부의 설계에 사용됨***

- □ 코딩 패턴은 특정한 프로그래밍 언어의 특징에 종속적일 수 있다.

❓ 다음 중 생성(creational) 패턴이 아닌 것은?

- □ Factory Method Pattern
- □ Abstract Factory Pattern
- □ Singleton Pattern
- ☑ ~~State Pattern~~

→ ***State Pattern은 행위(Behavioral) 패턴임***