Design Pattern

Research

**목 차**

[1 [product]\_duplex에 대한 설명 4](#_Toc532306978)

[1.1 개요 4](#_Toc532306979)

[1.2 구성 방식: 4](#_Toc532306980)

[1.3 사용법 5](#_Toc532306981)

[1.3.1 .conf 파일 설정 5](#_Toc532306982)

[2 동작 방식 7](#_Toc532306983)

[2.1 프로세스 감시 & 재시작 7](#_Toc532306984)

[2.1.1 프로세스 감시 – 공유 메모리 확인 7](#_Toc532306985)

[2.1.2 프로세스 감시 – pid 파일 확인 7](#_Toc532306986)

[2.2 열린 FD의 비율 검사 8](#_Toc532306987)

**그림 목차**

[Figure 1‑1 watcher의 조망 4](#_Toc532306988)

[Figure 2‑1 pid 파일의 내용 7](#_Toc532306989)

[Figure 2‑2 열린 fd 개수의 비율을 구하는 수식 및 명령어 8](#_Toc532306990)

[Figure 2‑3 열린 FD의 비율이 지정 비율을 넘을 경우 기록되는 시스템 로그 9](#_Toc532306991)

# 개요

디자인 패턴의 정의에 대해 설명합니다.

## Design Pattern?

(?)



* 이런 패턴을 뜻하는 게 아님
* 어떤 상황에서 ‘이런 거 쓰면 되게 좋더라’하는 방법들을 모아놓은 것이라 생각하면 됨
* 프로그램을 짜다 보니 다양한 클래스와 객체가 만들어지고 이 클래스들 간의 관계가 생성됨
  + 이 관계에 대한 정리 없이 중구난방으로 만든 프로그램은 유지보수 및 코드 분석에 많은 어려움을 겪음
  + 또한 구현 도중 요구사항의 변화가 생기면 이를 코드에 적용하는 데에도 많은 시간이 소모됨
* 그래서 많은 사람들이 이를 처리하는 방법을 고민
  + 프로그램 내부 클래스들이 갖는 구조에는 일정한 ‘형태’ 또는 ‘패턴’이 존재한다는 것을 알아차림
  + 객체지향적인 설계를 위한 노하우를 모아 클래스 설계 형태를 만들기 시작했고, 이를 디자인 패턴이라고 이름붙임

## 종류는?

* 방법의 목적과 형태에 따라 3가지로 구분. 대표적인 GoF(Gang of Four)를 기준으로 설명함

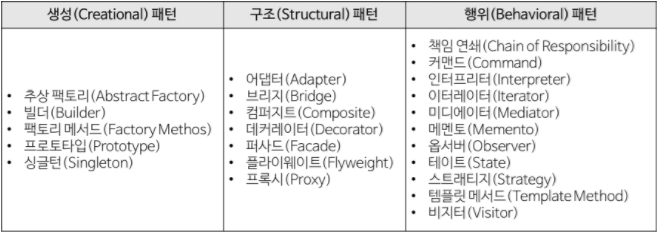


Figure 1‑1 GoF Design Pattern

### 생성 패턴 (Creational Pattern)

* **객체 생성**에 관련된 패턴
* 객체의생성과 조합을 캡슐화해 객체가 생성되고 변경되더라도 프로그램 구조의 영향을 받지 않도록 유연성 제공
* ex) 피자 가게가 있고 그 가게에서 사용자가 원하는 피자를 만들어줌

### 구조 패턴(Structural Pattern)

* 클래스나 객체를 조합해 **더 큰 구조**를 만드는 패턴
* 예를 들어 서로 다른 인터페이스를 지닌 2개의 객체를 묶어 단일 인터페이스를 제공하거나 객체들을 서로 묶어 새로운 기능을 제공하는 패턴
* //TODO 공부 이후 정의에 대한 상세 설명 추가 필요

### 행위 패턴 (Behavioral Pattern)

* 객체나 클래스 사이의 알고리즘이나 **책임 분배**에 관련된 패턴
* 한 객체가 혼자 수행할 수 없는 작업을 여러 개의 객체로 분배하면서 객체들 사이의 결합도를 최소화하는 것에 중점을 둔다
  + 즉, 거미줄처럼 얽혀있지 않도록 하는 것에 중점을 둔다는 것
* // TODO

# 생성 패턴

## 추상 팩토리 (Abstract Factory)

* // TODO

## 빌더 (Builder)

* // TODO

## 팩토리 메소드 (Factory Method)

* // TODO

## 프로토타입 (Prototype)

* // TODO

## 싱글톤 (Singleton)

* 프로그램의 라이프사이클 안에서 1개의 인스턴스만 생성할 수 있도록 허락하는 패턴
* 설정 파일의 값을 가져오는 기능이 필요한 경우
  + 이를 필요로 하는 클래스에 대해서 모두 설정 값 read하는 객체를 만드는 것은 비효율적
  + 1개만 만들고 race condition만 고려하면 자원을 아낄 수 있음
* 특징
  + Singleton 객체로 만들 클래스의 생성자는 private하다. 즉, 외부에서 객체 생성이 불가능하다.
  + getInstance 메소드에서 해당 클래스의 객체(자기 자신)을 갖고 있으며 이를 반환한다.
  + getInstance와 private instance는 static으로 만들어 외부에서는 객체 선언 없이 클래스 이름만으로 접근할 수 있게 해야 한다
  + 만약 여러 곳에서 접근이 필요한 객체라면, race condition을 고려하여 설계해야 한다. (객체는 1개뿐이므로)