디자인 패턴 과 MVVM

소프트웨어 생명주기(SDLC, Software Development Life Cycle)

소프트웨어 개발의 단계별 과정을 정의한 것

디자인 패턴

디자인 패턴은 이 생명주기의 특정 단계(주로 설계와 구현)에서 효율적이고 재사용 가능한 솔루션을 제공하기 위해 사용된다. 즉, 디자인 패턴은 SW의 생명주기를 구체적으로 구현하는 도구 또는 방법론의 일종이라고 할 수 있다.

SDLC 와 디자인 패턴이 어떤 식으로 연관되나?

1. 요구사항 분석

- 이 단계에서는 고객의 요구사항을 수집하고, 무엇을 개발해야 하는지 정의한다.
- 디자인 패턴의 역할:
    - 이 단계에서는 디자인 패턴이 직접 사용되지는 않지만, 어떤 아키텍처를 사용할지를 고려할 수 있다.
    - 예를 들어, “UI 가 중심인 애플리케이션으로 MVVM을 채택하자” 라는 결정을 할 수 있다.

2. 설계 단계

- 이 단계에서 시스템의 구조와 모듈을 설계한다.
- 디자인 패턴이 이 단계에서 가장 많이 활용된다.
- 디자인 패턴의 역할:
    - 아키텍처 패턴:
        - 애플리케이션의 큰 구조를 설계하는데 도움을 준다.
        - MVC,MVVM,MVP,Clean Architecture, Microservice등
    - 설계 패턴:
        - 각 모듈의 내부 설계를 구체화할 때 사용된다.
        - Singleton, Factory, Observer, Strategy, Builder 등
    - 이 단계에서 디자인 패턴은 코드 작성 전에 설계의 방향성을 제시하고, 코드의 구조적 문제를 미리 해결한다.

3. 구현 단계

- 설계된 구조를 바탕으로 실제 코드를 작성하는 단계이다.
- 디자인 패턴의 역할:
    - 구현 시 코드의 재사용성과 유지보수성을 높이기 위해 활용된다.
    - 예를 들어 객체 생성과 관리를 위해 Factory 패턴을 사용하거나, 데이터와 UI간의 동기화를 위해 Observer 패턴을 사용할 수 있다.
    - MVC, MVVM 같은 아키텍처 패턴이 코드 구조를 조직화하는데 도움을 준다.

4. 테스트 단계

- 구현된 코드를 검증하고 오류를 찾아 수정하는 단계이다.

- 디자인 패턴의 역할:
    - 설계와 구현 단계에서 사용된 디자인 패턴은 테스트를 용이하게 한다.

5. 배포 단계

- 소프트웨어를 실제 환경에 배포하는 단계이다.
- 디자인 패턴의 역할:
    - 이 단계에서는 디자인 패턴이 직접적으로 사용되지는 않지만, 설계와 구현에서 사용된 패턴이 유지보수를 용이하게 하여 배포 이후 문제를 빠르게 해결하는데 도움을 준다.

6. 운영 및 유지보수 단계

- 시스템이 운영되면서 수정하거나 기능을 추가하는 단계이다.
- 디자인 패턴의 역할:
    - 설계와 구현에서 사용된 패턴이 모듈화와 유연성을 제공하기 때문에 유지보수 작업이 용이하다.
    - 예를 들어, Decorator 패턴을 사용하면 기능을 쉽게 확장할 수 있고, Strategy 패턴을 사용하면 비즈니스 로직을 변경하기 쉬워진다.

→ 잘 설계된 패턴은 설계와 구현 단계에서 작업 효율성을 높이고, 소프트웨어의 품질을 보장한다.

→ 디자인 패턴은 SDLC를 효과적으로 진행하기 위한 도구라고 보면 된다.

MVC, MVVM 패턴 설명

소프트웨어 개발에서 사용자 인터페이스와 비즈니스 로직을 분리하기 위한 아키텍처 패턴이다. 각 패턴은 데이터(Model), 사용자 인터페이스(View), 그리고 비즈니스 로직(Controller/ ViewModel)을 나누는 방법과 상호작용 방식에서 차이가 있다.

MVC (Model-View-Controller)

가장 널리 사용되는 디자인 패턴으로, 애플리케이션의 구조를 세 가지 주요 컴포넌트로 나눈다.

구성 요소

- Model : 데이터와 비즈니스 로직
  - 데이터베이스와 상호작용하거나 데이터 처리를 담당
- View: 사용자 인터페이스(UI)
  - 사용자가 보는 화면
- Controller: 사용자 입력과 Model, View 간의 상호작용을 담당한다.
  - 사용자의 요청을 받아 Model에 전달하고, 결과를 View에 전달한다.

데이터 흐름

1. 사용자가 View와 상호작용(예: 버튼 클릭)
2. Controller가 사용자 입력을 처리.
3. Controller가 Model을 호출하여 데이터를 변경하거나 가져옴
4. Model이 데이터를 처리하고 ,Controller에 전달
5. Controller가 View에 결과를 전달해 UI를 업데이트

장점

- View와 Model의 분리가 명확하여 재사용성과 유지보수가 용이.
- 다양한 View에서 동일한 Model을 재사용 가능

단점

- Controller가 복잡한 어플리케이션에서는 비대해질 수 있음

MVVM (Model-View-ViewModel)

MVVM은 주로 UI 중심의 프레임워크(WPF, SwiftUI) 등에서 사용된다.

데이터 바인딩을 통해 View와 ViewModel 간의 실시간 동기화를 지원한다.

구성 요소

- Model: 데이터와 비즈니스 로직
- View: 사용자 인터페이스(UI)
    - ViewModel과 데이터를 바인딩하여 화면을 자동으로 갱신
- ViewModel: View와 Model 간의 중간 관리자 역할
    - 데이터를 가공해 View에 제공하고, 사용자 입력을 Model로 전달

데이터 흐름

1. 사용자가 View와 상호작용
2. View가 ViewModel에 데이터를 바인딩
3. ViewModel이 Model과 상호작용하여 데이터를 처리
4. Model의 데이터 변경이 ViewModel에 반영
5. ViewModel이 View를 자동으로 업데이트

장점

- 데이터 바인딩을 통해 View와 ViewModel 간의 실시간 동기화가 가능
- View와 Model 간의 결합도가 낮아 재사용성이 높음