[소개 페이지]

프로젝트명: java_design_patterns

1. 목적:

이 프로젝트는 Java에서 자주 사용되는 디자인 패턴을 학습하고 활용할 수 있도록 간단한 예제와 설명을 제공하기 위해 만들어졌습니다. 이를 통해 개발자들은 코드의 유지보수성, 확장성, 가독성을 향상시킬 수 있습니다.

2. 주요 내용:

1) 생성 패턴 (Creational Patterns)

객체 생성 방식을 다루며, 상황에 맞는 객체 생성 방법을 제공합니다.

싱글톤 (Singleton): 클래스의 인스턴스가 하나만 존재하도록 보장하고, 전역적으로 접근할 수 있도록 합니다.

팩토리 (Factory): 객체 생성 로직을 클라이언트로부터 숨기고, 필요한 객체를 생성합니다.

빌더 (Builder): 복잡한 객체를 단계적으로 생성할 수 있도록 도와줍니다.

2) 구조 패턴 (Structural Patterns)

클래스와 객체를 조합하여 더 큰 구조를 만드는 방법을 다룹니다.

어댑터 (Adapter): 서로 호환되지 않는 인터페이스를 연결합니다.

컴포지트 (Composite): 개별 객체와 그 조합을 동일하게 취급합니다.

프록시 (Proxy): 다른 객체에 대한 대리자 또는 대체 객체를 제공합니다.

3) 행동 패턴 (Behavioral Patterns)

객체 간의 알고리즘과 상호작용 방식을 다룹니다.

옵저버 (Observer): 객체 상태 변경 시 관련 객체들에게 자동으로 알림을 제공합니다.

전략 (Strategy): 알고리즘을 캡슐화하여 교체 가능하도록 만듭니다.

커맨드 (Command): 요청을 객체로 캡슐화하여 매개변수화 및 요청 저장을 가능하게 합니다.

디자인 패턴의 주요 범주를 다루며, 각 패턴의 특징 및 사용 사례를 코드와 함께 설명합니다.

3. 활용 방법:

개발자는 이 레포지토리를 통해 다음을 학습할 수 있습니다

- 패턴을 설계에 통합하여 소프트웨어 품질을 높이는 방법

- 객체지향 설계 원칙을 준수하면서 문제를 해결하는 전략

- 코드 재사용성과 확장성을 고려한 설계

[활용 예시]

1. 실무 프로젝트에서의 활용

프로젝트를 설계할 때 디자인 패턴을 도입하면 코드의 구조를 개선하고 유지보수성을 높일 수 있습니다.

예시)

문제: 웹 애플리케이션에서 사용자 요청에 따라 다른 유형의 알림(이메일, 푸시, SMS)을 전송해야 한다.

해결: 레포지토리에서 제공하는 팩토리 메서드 패턴 예제를 활용하여 각 알림 유형을 생성하는 팩토리 클래스를 구현할 수 있습니다. 이를 통해 객체 생성 로직을 분리하고, 새로운 알림 유형 추가 시 최소한의 수정으로 대응할 수 있습니다.

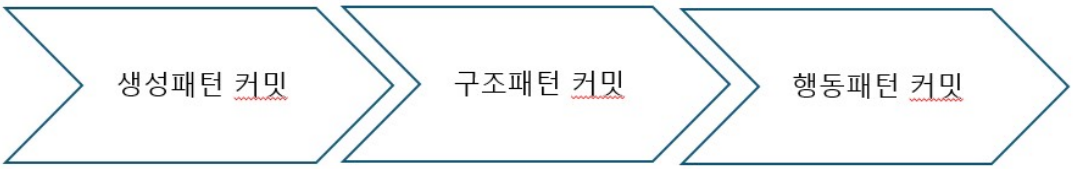
2. 팀 협업에서의 활용

레포지토리를 활용하면 팀 내 설계 원칙을 통일하고 협업 효율성을 높일 수 있습니다.

팀 교육 자료로 활용: 팀원들이 패턴을 쉽게 이해하고 적용할 수 있도록 예제를 공유

통일된 설계 기준 제공: 프로젝트에서 동일한 패턴을 사용해 일관성 있는 코드 작성

코드 리뷰 자료: 레포지토리의 구현 방식을 참고해 팀의 코드를 리뷰하고 개선점을 찾음



로드맵