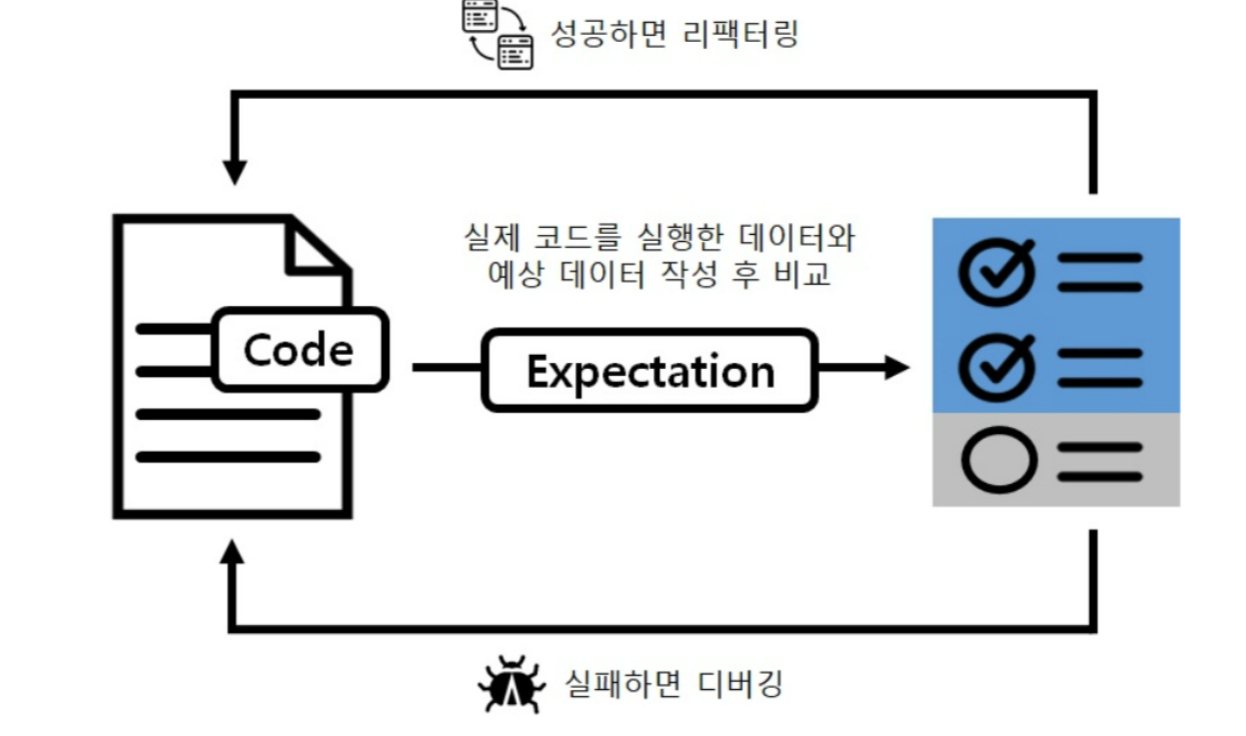
테스트 코드 작성하기

Test 실행을 통해 다양한 문제를 미리 예방하고 코드 변경 등으로 인해 발생하는 부작용 방지 가능

1. 예상 데이터 작성하기
2. 실제 데이터 획득하기
3. 예상 데이터와 실제 데이터 비교해 검증하기



테스트 코드는 다양한 경우를 대비해 테스트 케이스를 작성한다.

테스트를 통한 코드 검증과 리팩터링은 개발자의 기본 소양이기에 테스트 주도 개발(TDD) 방식을 채택 -> 테스트 코드를 만든 후 이를 통과하는 최소한의 코드부터 시작해 점진적으로 코드를 개선 및 확장해 나가는 개발 방식

테스트 케이스 작성법

메소드에 우클릭 후 제너레이트 -> test 실행



show() 테스트 진행하기

테스트 케이스 : 프로그램의 다양한 상황을 자동으로 검증하기 위한 테스트 코드

디버깅 : 프로그램에서 발생한 문제의 원일을 찾고 고치는 작업

리팩토링 : 프로그램의 수행 결과는 그대로 유지하면서 코드의 구조 및 성능을 개선

@SpringBootTest : 스프링 부트 환경과 연동된 테스트를 위한 어노테이션

@Transactional : 테스트를 수행해 조작된 데이터를 복구하는 어노테이션

마무리

1. 테스트 : 프로그램의 품질을 검증하는 것으로, 의도대로 프로그램이 잘 동작하는지 확인하는 과정

2. 테스트 코드 작성법

1. 예상데이터 작성하기

2. 실제 데이터 획득하기

3. 예상 데이터와 실제 데이터 비교해 검증하기의

프로그램 코드가 테스트를 통과하면 지속적인 리팩토링을 진행하고, 통과하지 못한다면 잘못된 부분을 찾아 고치는 디버깅 진행

3. 테스트 케이스

테스트는 다양한 경우를 대비해 작성하는데, 이를 테스트 케이스라고 한다.

4. test 디렉토리 위치

test 디렉토리는 main 디렉토리 아래에 만들어지며, 기존 자바 코드가 담긴 main > java 디렉토리와 데칼코마니처럼 test > java 디렉토리로 생성

5. @SpringBootTest

스프링부트와 해당 클래스를 연동해 테스트 할때 사용하는 어노테이션으로 테스트 코드에서 스프링부트가 관리하는 다양한 객체 주입이 가능

6. @Test

해당 메소드가 테스트를 위한 코드라고 선언하는 어노테이션을 의미

7. 테스트 케이스의 트랜잭션 처리

테스트 시 데이터를 조회하는 것이 아닌 조작(생성 / 수정 / 삭제)하는 작업을 했다면 테스트를 마치고 데이터를 롤백하는 과정을 진행하여야함. 롤백은 테스트 메서드 위에 @Transactional 어노테이션을 붙여 해당 메서드를 트랜잭션 처리해주면 된다.