TDD 시작

TDD란?

TDD는 테스트부터 시작한다. 테스트를 먼저하고 그 다음에 구현한다는 것이다.

구현 코드가 없는데 어떻게 테스트를 할까?
여기서 테스트를 먼저 한다는 것은 테스트 코드를 작성한다는 것을 의미한다.

테스트 코드를 먼저 작성함으로써 다음과 같은 고민을 하게 된다.

- 테스트 대상이 될 클래스이름과 메서드 이름
- 매개변수의 개수
- 리턴 타입
- 인스턴스 메서드로 구현할지, static 메서드로 구현할지

@Test는 해당 애너테이션이 붙은 메서드를 테스트 메서드로 인식한다.

assertEquals( ) 메서드는 인자로 받은 두 값이 동일한지 비교한다.

TDD의 흐름 : Red-Green-Refactor

TDD는 테스트 코드를 작성하고 테스트에 실패하면 테스트를 통과시킬 만큼의 코드를 추가한다.
테스트를 통과한 뒤에는 개선할 코드가 있다면 리팩토링한다.
다시 테스트를 실행하여 기존 기능이 망가지지 않았는지 확인한다.

이러한 과정을 반복하면서 점진적으로 기능을 완성해 나간다. 이것이 전형적인 TDD의 흐름이다.

> 💡 **Red-Green-Refactor**
> 이러한 TDD의 흐름을 Red-Green-Refactor라고도 하는 데,
> 여기서 레드는 실패한 테스트, 그린은 성공한 테스트, 리팩터는 리팩토링 과정을 의미한다.

TDD 실습 : 암호 강도 측정기

네 번째 테스트 : 값이 없는 경우

테스트 코드를 작성하는 과정에서 값이 없는 경우를 테스트하는 것이 중요하다.

암호 강도 측정기는 null을 입력할 경우 어떻게 반응해야 할까?
책의 저자는 다음과 같은 두 가지 방법을 떠올렸다고 한다.

1. illegalArgumentException을 발생시킨다.
2. 유효하지 않은 암호를 의미하는 PasswordStrength.INVALID를 리턴한다.

TDD의 장점

테스트 코드를 먼저 작성하는 TDD의 흐름은 조금은 낯설어 보인다.
그럼에도 이러한 TDD를 하는 이유는 무엇일까?

1. 개발 범위를 확정

테스트 코드를 먼저 작성할 때, 가장 먼저 통과해야 할 테스트를 작성한다.

이러한 테스트를 통과할 코드만을 작성하기 때문에 테스트 코드를 작성함으로써
개발의 범위가 정해진다.

테스트 코드가 추가되면서 검증하는 범위가 넓어지고 이로 인해 구현이 완성되어 간다.

2. 지속적인 리팩토링

잘 동작하는 코드를 수정하는 것은 심리적으로 불안감을 준다.

하지만 해당 기능이 온전하게 동작한다는 것을 검증해주는 테스트가 있으면
코드 수정의 심리적 불안감을 줄여주고 결국 리팩토링을 보다 과감하게 진행할 수 있게 된

다.

이러한 지속적인 리팩토링은 개발 과정에서 코드의 품질이 급격하게 나빠지는 것을 방지할 수 있으며 향후 유지보수 비용을 낮추는데 기여한다.