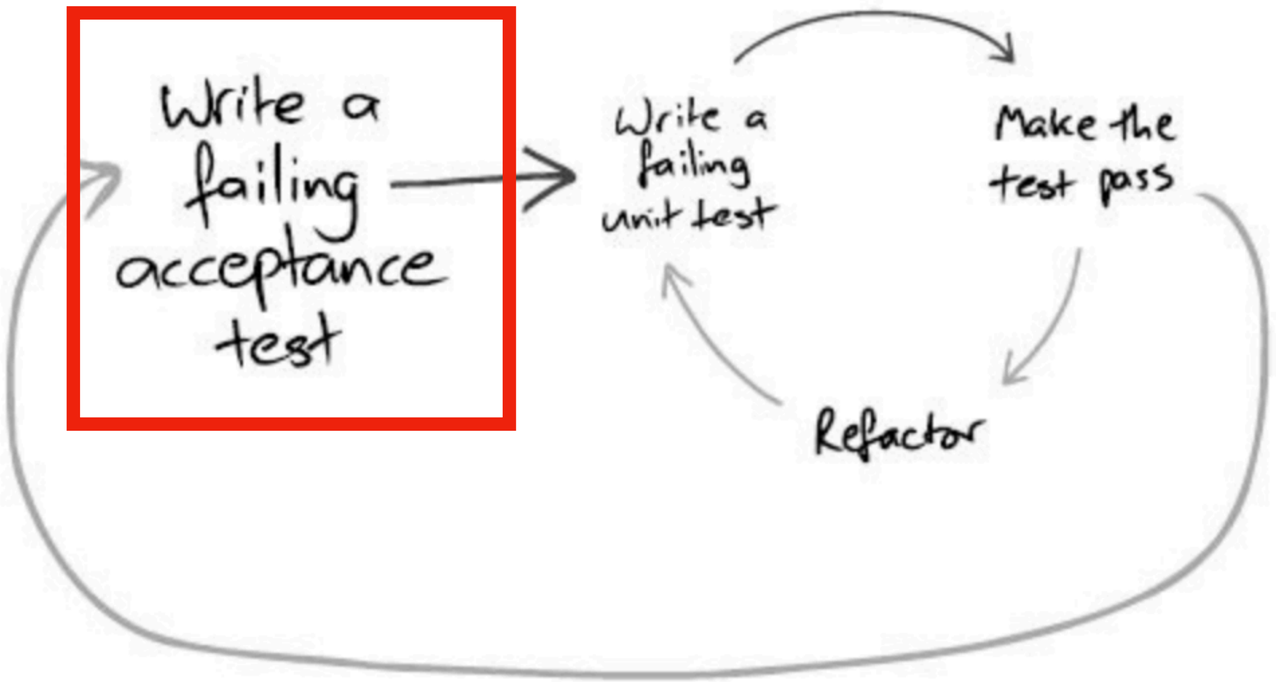
3주차 피드백 및 다시보기

다시 보기

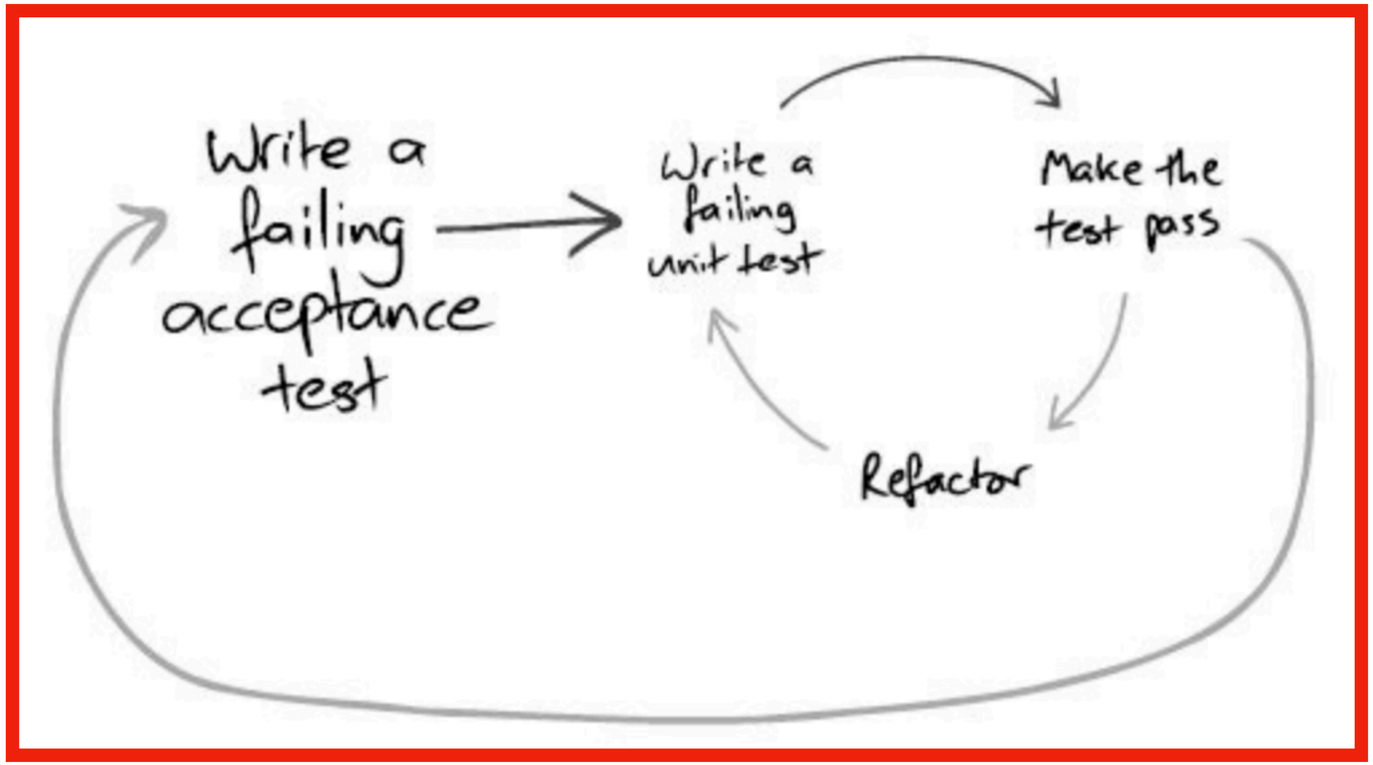
3주차

- 인수 테스트

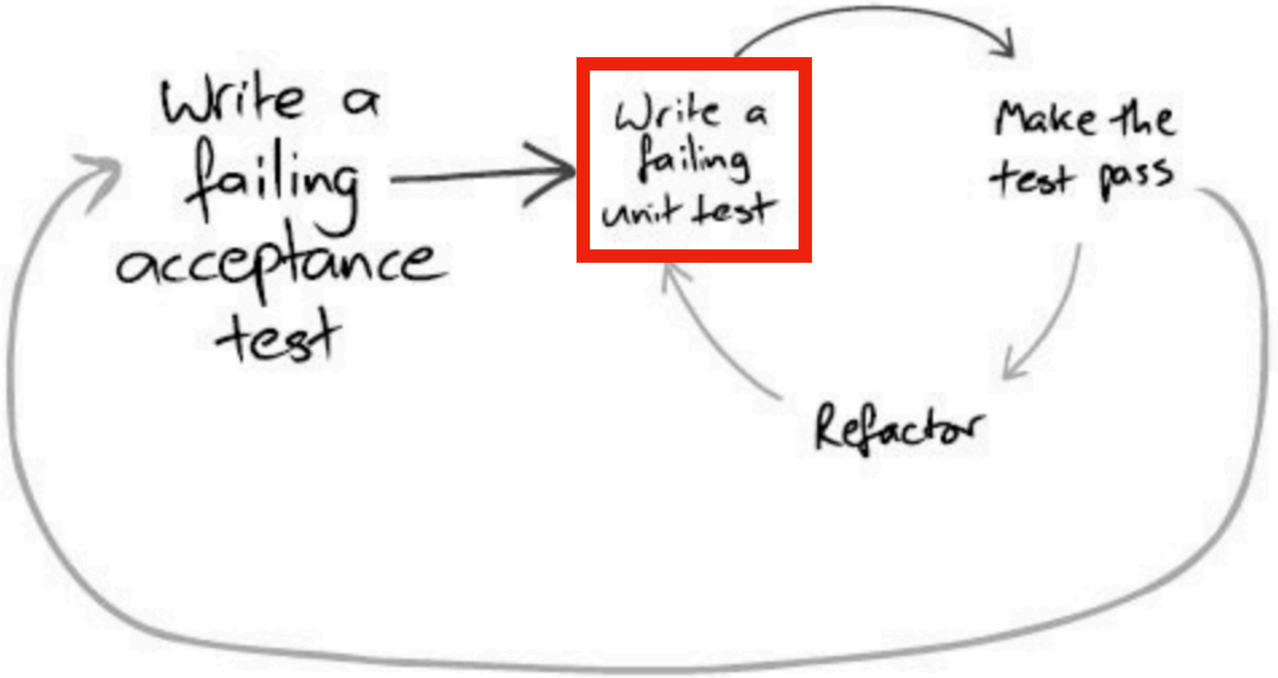

Write a failing acceptance test
Write a failing unit test
Make the test pass
Refactor

5주차

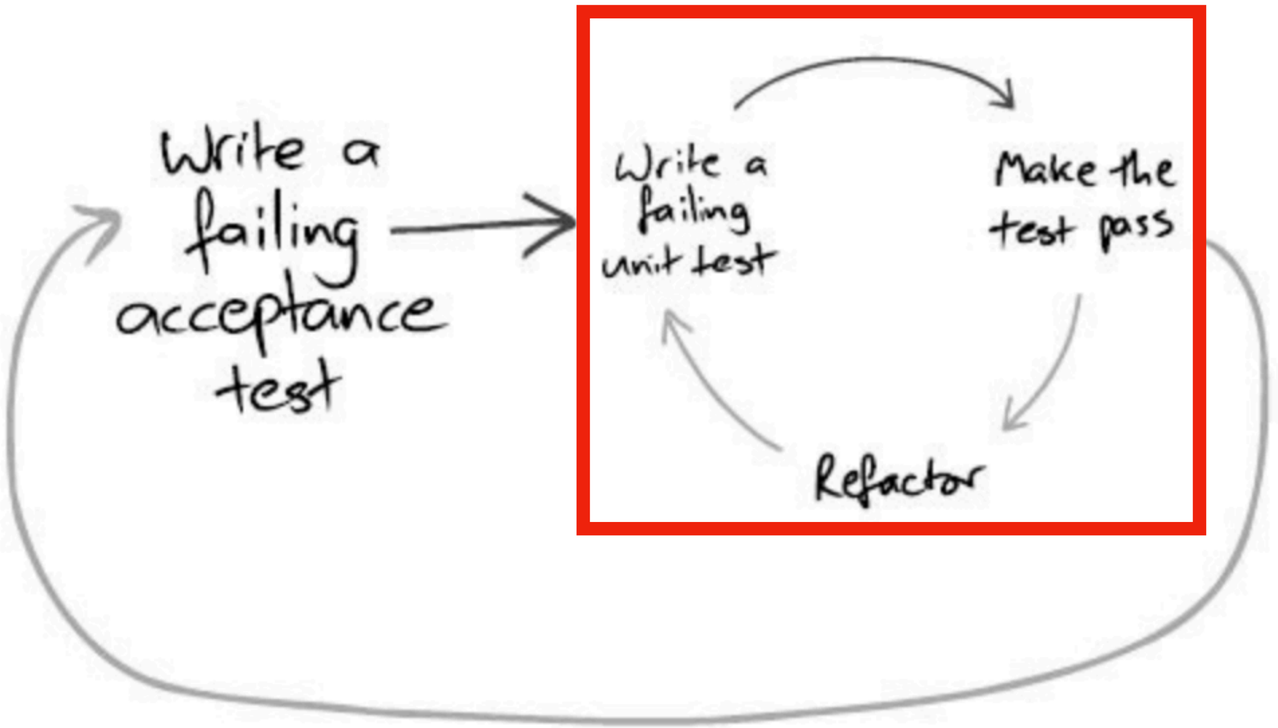
- 인수 테스트를 이용한 개발 흐름



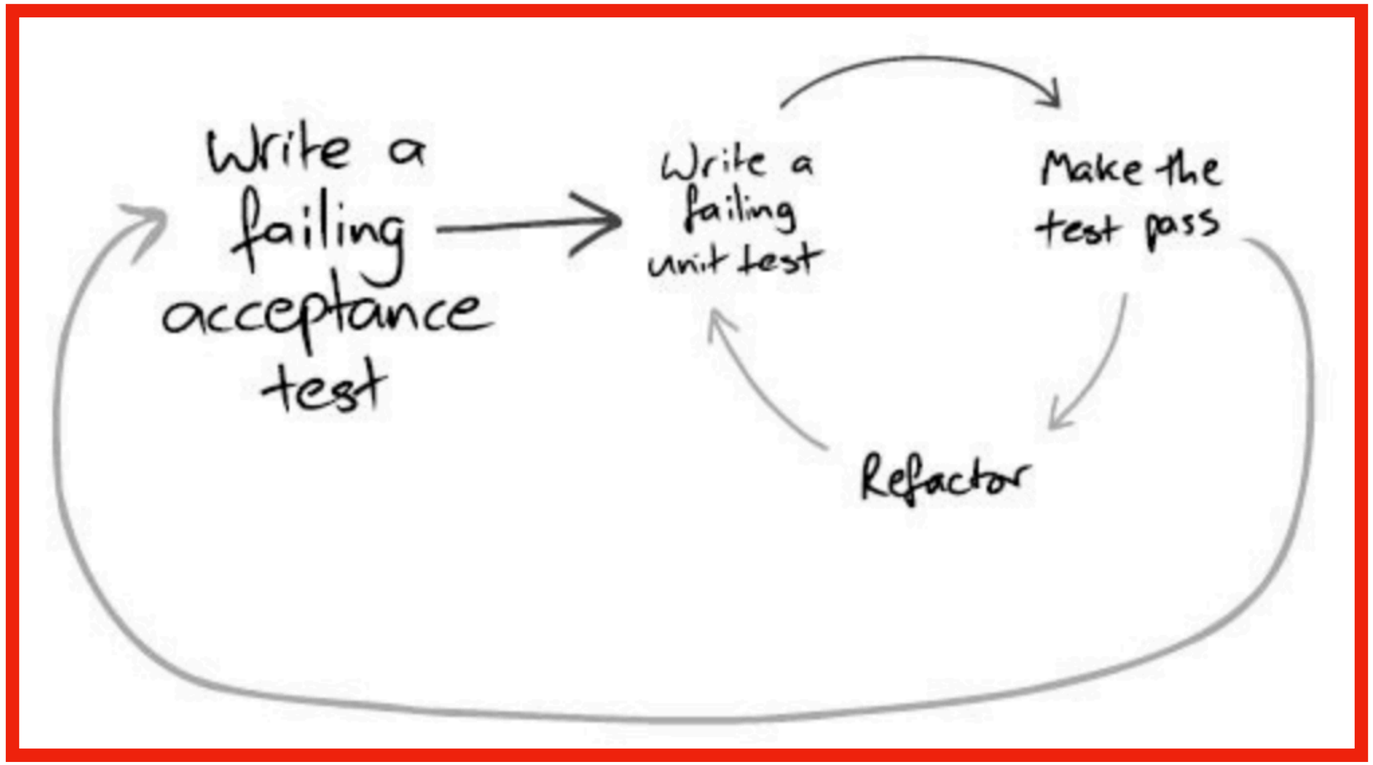
테스트


Write a failing acceptance test
Write a failing unit test
Make the test pass
Refactor

테스트 주도 개발 사이클


Write a failing acceptance test
Write a failing unit test
Make the test pass
Refactor

인수 테스트 주도 개발 사이클



인수 테스트 격리

테스트 격리

- @DirtiesContext
- Table Truncate

@DirtiesContext

- 효과적인 테스트 수행을 위해 스프링에서는 context caching 기능 지원
- @DirtiesContext를 활용하여 캐시 기능을 사용하지 않게 설정
- 매번 Context를 새로 구성하다보니 시간이 많이 걸림

DatabaseCleanup

- EntityManager를 활용하여 테이블 이름 조회 후 각 테이블 Truncate 수행
- ID auto increment 숫자를 1로 복구 시킴

인수 테스트 가독성

테스트의 의도를 명확히 드러내기

```java
@DisplayName("지하철 노선을 조회한다.")
@Test
void getLine() {
    // given
    // 지하철_노선_등록되어_있음
    Map<String, String> params = new HashMap<>();
    params.put("name", "신분당선");
    params.put("color", "bg-red-600");
    params.put("startTime", LocalTime.of( hour: 05,  minute: 30).format(DateTimeFormatter.ISO_TIME));
    params.put("endTime", LocalTime.of( hour: 23,  minute: 30).format(DateTimeFormatter.ISO_TIME));
    params.put("intervalTime", "5");

    ExtractableResponse<Response> createResponse = RestAssured.given().log().all().
            contentType(MediaType.APPLICATION_JSON_VALUE).
            body(params).
            when().
            post( path: "/lines").
            then().
            log().all().
            extract();

    // when
    // 지하철_노선_조회_요청
    String uri = createResponse.header( name: "Location");

    ExtractableResponse<Response> response = RestAssured.given().log().all().
            accept(MediaType.APPLICATION_JSON_VALUE).
            when().
            get(uri).
            then().
            log().all().
            extract();

    // then
    // 지하철_노선_응답됨
    assertThat(response.statusCode()).isEqualTo(HttpStatus.OK.value());
    assertThat(response.as(LineResponse.class)).isNotNull();
}
```

```java
@DisplayName("지하철 노선을 조회한다.")
@Test
void getLine() {
    // given
    ExtractableResponse<Response> createResponse = 지하철_노선_등록되어_있음( name: "신분당선",  color: "RED");

    // when
    ExtractableResponse<Response> response = 지하철_노선_조회_요청(createResponse);

    // then
    지하철_노선_응답됨(response, createResponse);
}
```

테스트 가독성이 중요한 이유

- 가독성이 좋지 않으면 방치되는 테스트가 될 가능성이 높다
  - @Ignore or @Disabled
- 변경 사항에 대해서 수정하기 어렵다. -> 방치될 가능성 높다
  - @Ignore or @Disabled
- 가독성이 좋으면 해당 기능의 스펙을 나타낼 수 있다.

프로덕션 코드의 가독성이 중요한 만큼 테스트 코드의 가독성도 중요함

테스트 코드 중복 제거

- 기능 개발 간 테스트 코드도 중복이 많이 발생함
- 테스트 가독성을 저해하는 구조가 나올 수 있어 중복제거가 중요함
- 가독성이 좋지 않은 테스트 코드는 관리가 되지 않는 가능성이 높음

중복 제거 방법

- 메서드 분리
- CRUD 추상화
- Cucumber나 JBehave 와 같은 BDD 도구 사용

메서드 분리

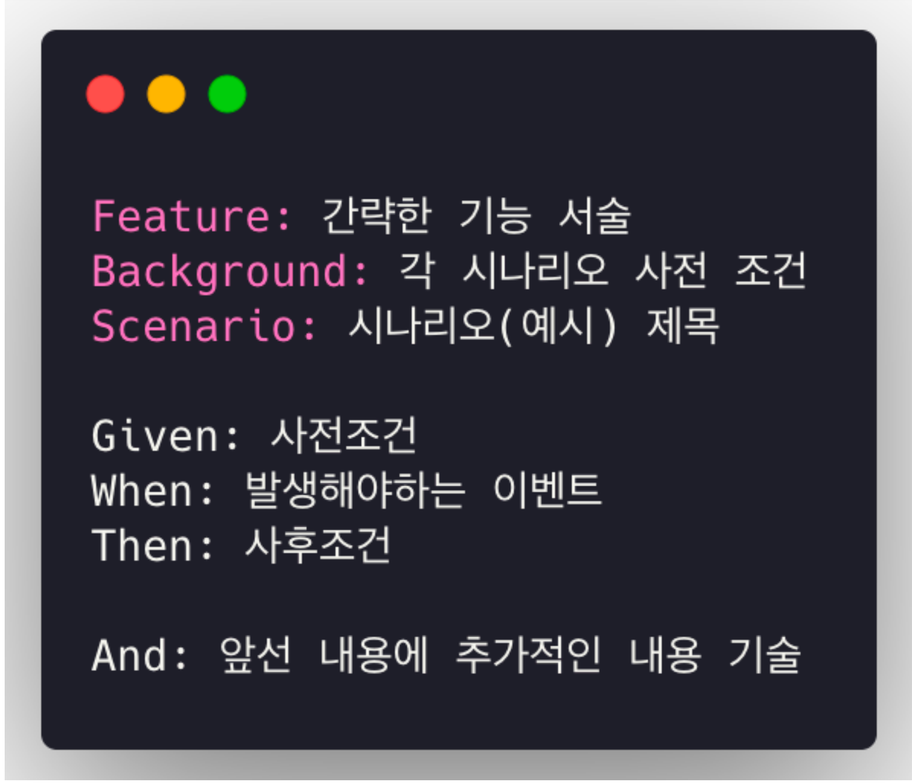
- 반복되는 코드는 메서드로 분리
  - ex) StationAcceptanceTest

의도 드러내기

- 테스트의 각 스텝을 한글 메서드로 분류하기
  - ex) 각 요청에 대한 응답

인수 조건, 인수 테스트 작성 팁

인수 조건을 테스트로 옮기기



인수 조건 예시

```
Feature: 지하철 역 관리 기능

  Scenario: 지하철 역을 생성한다.
    When 지하철 역을 생성 요청한다.
    Then 지하철역이 생성된다.

  Scenario: 지하철 역을 삭제한다.
    Given 지하철 역이 등록되어있다.
    When 지하철 역을 삭제 요청한다.
    Then 지하철 역이 삭제된다.
```

ex) 지하철 역 등록

- when: 지하철 역 등록 request 보내기
- then: 지하철 역 등록을 확인하기
  - 방법1: 지하철 역 요청의 응답 코드로 확인
  - 방법2: 지하철 역 조회 request 후 response 값에서 확인

ex) 지하철 역 삭제

- given: 지하철 역 등록 request 보내기
- when: 지하철 역 삭제 request 보내기
- then: 지하철 역 삭제를 확인하기
  - 방법1: 지하철 역 요청의 응답 코드로 확인
  - 방법2: 지하철 역 조회 request 후 response 값에서 확인

인수 테스트 클래스

- Feature 기준으로 인수 테스트 클래스를 나눌 수 있음
- Scenario 기준으로 인수 테스트 메서드를 작성할 수 있음 하나의
- Feature 내부에 있는 Scenario는 같은 테스트 픽스쳐를 공유하기

간단한 성공 케이스 우선 작성

- 여기서 말하는 간단한은 지나치게 간단한이 아님
- 동작 가능한 가장 간단한 성공 케이스로 시작
- 테스트가 동작하면 실제 구조에 관해 더 좋은 생각이 떠오를 수 있음
- 그 과정에서 발생 가능한 실패를 처리하는것과 성공 케이스 사이에서 우선 순위를 가늠

실패하는 테스트 지켜보기

- 코드를 작성하기 전 테스트가 실패하는 것을 지켜본 후 진단 메시지를 확인
- 테스트가 예상과 다른 식으로 실패하면 뭔가를 잘못 이해했거나 코드가 완성되지 않았다는 뜻

Given / When / Then

- When -> Then -> Given 순서로 작성하기