다트 테스트 케이스 작성하기

테스트 기반 개발의 이해와 실습

학습목표

- 다트 테스트 프레임워크의 기본 개념과 구조 이해
- 단위 테스트와 통합 테스트의 작성 방법 습득
- AAA (Arrange, Act, Assert) 패턴을 활용한 테스트 케이스 작성법 숙지

다트 테스트의 기본 구조

- 테스트는 단위(Unit), 통합(Integration), 위젯 UI 테스트로 구분
- 테스트 프레임워크는 다트와 플러터에서 공통으로 사용
- 개발된 코드의 요구사항 충족 여부를 자동으로 검증
- 코드 변경 시 기존 기능 손상 여부를 빠르게 확인 가능

테스트 환경 구성

- pubspec.yaml 파일에 test 패키지를 dev_dependencies로 추가
- 테스트 코드는 test 하위 폴더에 위치
- DartPad 사용 시 flutter_test 패키지 import 필요
- IDE 사용 시 코드와 테스트를 별도 파일로 분리 가능

Travel.dart

```dart
const convertToKilometers = 1.60934; // 마일을 킬로미터로 변환하는 상수
const convertToMiles      = 0.62137119; // 킬로미터를 마일로 변환하는 상수

class Travel {
  late double distance;

  Travel(double newDistance) {
    distance = newDistance;
  }

  double distanceToMiles() {
    return distance * convertToMiles;
  }

  double distanceToKilometers() {
    return distance * convertToKilometers;
  }
}
```

test/travel_test.dart

```dart
import 'package:test_dart_sample/test_dart_sample.dart'; // 테스트할 코드가 있는 라이브러리 import
import 'package:test/test.dart'; // 다트 테스트 패키지 import

void main() {
  test('Travel Distance', () { // 'Travel Distance'라는 이름의 테스트 케이스 정의
    // Arrange: 테스트에 필요한 변수 및 객체 설정
    var distance = 10.0; // 테스트할 거리 값
    var expectedDistance = distance; // 예상되는 거리 값

    // Act: 실제 테스트 대상 코드 실행
    var travel = Travel(expectedDistance); // Travel 객체 생성
    var result = travel.distance; // Travel 객체의 distance 속성 값 가져오기

    // Assert: 테스트 결과 검증
    expect(expectedDistance, result); // 예상 값과 실제 값이 일치하는지 확인
  });
}
```

test/travel_multiple_test.dart

```dart
import 'package:test_dart_sample/test_dart_sample.dart'; // 테스트할 코드가 있는 라이브러리 import
import 'package:test/test.dart'; // 다트 테스트 패키지 import

void main() {
  test('Travel Distance to Miles', () { // 마일 변환 테스트
    // Arrange
    var miles = 10.0; // 마일 값
    var expectedMiles = miles * convertToMiles; // 예상되는 마일 값 (변환 상수 적용)

    // Act
    var travel = Travel(miles); // Travel 객체 생성
    var result = travel.distanceToMiles(); // 마일로 변환하는 메서드 호출

    // Assert
    expect(expectedMiles, result); // 예상 값과 실제 값이 일치하는지 확인
  });

  test('Travel Distance to Kilometers', () { // 킬로미터 변환 테스트
    // Arrange
    var kilometers = 10.0; // 킬로미터 값
    var expectedKiloMeters = kilometers * convertToKilometers; // 예상되는 킬로미터 값 (변환 상수 적용)

    // Act
    var travel = Travel(kilometers); // Travel 객체 생성
    var result = travel.distanceToKilometers(); // 킬로미터로 변환하는 메서드 호출

    // Assert
    expect(expectedKiloMeters, result); // 예상 값과 실제 값이 일치하는지 확인
  });
}
```

다트 테스트 패키지 애플리케이션에 추가하기

- `pubspec.yaml` 파일에 `test` 패키지를 `dev_dependencies`에 추가
- `dart pub add test --dev` 명령 사용
- 개발 단계에서만 필요함을 명시하기 위해 `--dev` 플래그 사용

샘플 테스트 애플리케이션 만들기

- 테스트 대상 클래스 및 함수를 포함하는 다트 파일 생성
- 예제에서는 `Travel` 클래스를 사용

다트 애플리케이션에서 단위 테스트 실행하기

- 테스트는 AAA (Arrange, Act, Assert) 패턴을 따름
  - Arrange: 테스트에 필요한 데이터 설정
  - Act: 테스트 대상 코드 실행
  - Assert: 결과 검증

요약

- 다트 테스트 프레임워크를 사용하여 테스트 케이스 작성 방법을 학습
- AAA 패턴을 따라 테스트 케이스를 작성하고 실행하는 방법을 익힘
- 테스트 케이스를 통해 코드의 정확성을 검증하고 유지보수성을 향상시킬 수 있음

END