Kotlin을 이용한 테스트 코드 작성

요약

- JUnit
  - 기존 풍부한 자바 기반 레퍼런스 참고 가능
  - 자바 방식의 코드 작성
    - 작성 코드 증가(가독성 감소) 및 DSL 스타일 코드 사용 불가
  - 번거로운 BDD(given-when-then 패턴) 테스트
- Kotest
  - 코틀린 방식의 코드 작성
    - 간결, 반복/중복 코드 제거, DSL 스타일 코드 사용 가능
  - Junit과 혼용 가능하나 디펜던시 추가 필요
  - 팀에 맞는 스타일 적용 가능 – Junit에서 대치 시 러닝 커브 존재

JUnit5을 이용한 단위 테스트

- 역따옴표(`)로 함수 이름을 한글과 공백으로 표현
- assertThrows, assertDoesNotThrows 같은 Kotlin 함수 이용

```
// Java 스타일
@DisplayName("회원의 비밀번호와 다를 경우 예외가 발생한다")
@Test
fun authenticate() {
    val user = createUser()
    assertThatExceptionOfType(UnidentifiedUserException::class.java)
        .isThrownBy { user.authenticate(WRONG_PASSWORD) }
}
```

```
// Kotlin 스타일
@Test
fun `회원의 비밀번호와 다를 경우 예외가 발생한다`() {
    val user = createUser()
    assertThrows<UnidentifiedUserException> {
        user.authenticate(WRONG_PASSWORD)
    }
}
```

테스트 팩토리

- 기본 인자와 이름 붙인 인자를 활용하여 테스트하려는 부분(관심사) 특정
- 자바 빌더 패턴 대체

```
createMission(submittable = true) // 과제 제출물을 제출할 수 있는 과제
createMission(submittable = false) // 과제 제출물을 제출할 수 없는 과제
createMission(title = "과제1", evaluationId = 1L) // 특정 평가에 대한 과제
createMission(hidden = false) // 숨기지 않은 과제

fun createMission(
    title: String = MISSION_TITLE,
    description: String = MISSION_DESCRIPTION,
    evaluationId: Long = 1L,
    startDateTime: LocalDateTime = START_DATE_TIME,
    endDateTime: LocalDateTime = END_DATE_TIME,
    submittable: Boolean = true,
    hidden: Boolean = false,
    id: Long = 0L
): Mission {
    return Mission(title, description, evaluationId, startDateTime, endDateTime, submittable,
}
```

테스트 확장 함수

- 검증하고자 하는 조건은 확장 함수 형태로 분리하여 가독성 향상

```
private fun ApplicantAndFormResponse.hasKeywordInNameOrEmail(keyword: String): Boolean {
    return name.contains(keyword) || email.contains(keyword)
}

@Test
fun `모집에 지원한 지원 정보를 특정 키워드로 조회하면 이름이나 이메일에 해당 키워드가 포함된 지원자
    val recruitmentId = 1L
    val keyword = "amazzi"
    val actual = applicantService.findAllByRecruitmentIdAndKeyword(recruitmentId, keyword)
    assertThat(actual).allMatch {
        it.hasKeywordInNameOrEmail(keyword)
    }
}
```

테스트 어설션

- 소프트 어설션: 일부 어설션 실패해도 테스트 계속 진행
- 객체 비교에서 각 프로퍼티를 따로따로 소프트 어설션하기 보다는 데이터 클래스로 한 번에 비교
  -> (실패한 원인이 되는 프로퍼티 쉽게 파악)

```
val expected = Speaker("Jason", "Mraz", 45)
val actual = Speaker("Jason", "Park", 29)
assertThat(actual).isEqualTo(expected)


expected: Speaker(firstName=Jason, lastName=Mraz, age=45)
 but was: Speaker(firstName=Jason, lastName=Park, age=29)
```

Kotest

- Junit5와 혼용 가능
- 스마트 캐스트 지원
- infix 함수로 간편한 표기

```
class MyTests : StringSpec({
    "length should return size of string" {
        "hello".length shouldBe 5
    }
    "startsWith should test for a prefix" {
        "world" should startWith("wor")
    }
})
```

```
val actual = userRepository.findByEmail("jason@woowahan.com")
actual.shouldNotBeNull() // 스마트 캐스트
actual.name shouldBe "박재성"
```

Kotest를 이용한 단위 테스트

- 테스트는 그냥 함수
- 테스트 중첩 가능
- 런타임에 테스트 함수 동적 생성 가능 -> 반복 항목 처리 수월
- 실행 중 예외 발생 시 프레임워크가 자동 실패 처리

```
class DynamicTests : FunSpec({

    listOf(
      "sam",
      "pam",
      "tim",
    ).forEach {
       test("$it should be a three letter name") {
           it.shouldHaveLength(3)
       }
    }
})
```

Kotest를 이용한 단위 테스트

- DSL 스타일 사용 가능

```
"아이디로 유저 조회 시 유저가 반환된다" {
    val expectedUser = User(name = "kyunam")
    every { userRepository.findById(1L) } answers { expectedUser }

    val searchedUser = userService.getById(1L)

    searchedUser.name shouldBe expectedUser.name
}

"1과 2를 더하면 3이 반환된다" {
    val result = sut.calculate("1 + 2")

    result shouldBe 3
}
```

Target platform: JVM Running on kotlin v. 1.5.0

Kotest 어설션

- 적합한 스타일 선택 사용 가능(StringSpec, FuncSpec, ...)

```
// JUnit 5
class AuthenticationCodeTest {
    @Test
    fun `인증 코드를 생성한다`() {
        val authenticationCode = AuthenticationCode(EMAIL)
        assertAll(
            { assertThat(authenticationCode.code).isNotNull() },
            { assertThat(authenticationCode.authenticated).isFalse()
            { assertThat(authenticationCode.createdDateTime).isNot
        )
    }
}
```

```
// Kotest
class AuthenticationCodeTest : StringSpec({
    "인증 코드를 생성한다" {
        val authenticationCode = AuthenticationCode(EMAIL)
        assertSoftly(authenticationCode) {
            code.shouldNotBeNull()
            authenticated.shouldBeFalse()
            createdDateTime.shouldNotBeNull()
        }
    }
```