آزمون نرم افزار

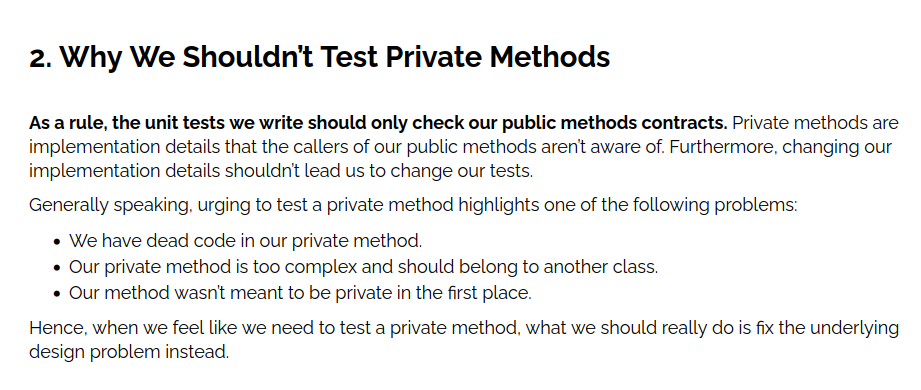
تمرین شماره 1 30 مهر 1402

محمدرضا بخشایش - 810199381

محمد امین باهنر – 810199548

سوال 1)

اکثریت مخالف تست کردن متدهای پرایوت هستند برای مثال در سایت <https://www.baeldung.com/java-unit-test-private-methods> به خوبی به دلایل این کار اشاره کرده است که چرا نباید تست کنیم



اما اگر اصرار بر این کار داریم میتوانیم از با استفاده از فرستادن یک سری متغییر درون متد هایی که از آن متد های پرایوت استفاده میکنند و دیدن اثر تغییرات آن متوجه درست بود متد های پرایوت بشویم.(reflection)

سوال 2)

بله میتوان اما چاش های جدی داریم که باید به آن‌ها توجه کنیم مانند مسأله race condition که میتوان با استفاده از mock و سایر ترفند ها این مسائل را حل کرد.

سوال3)

نکته کلی اینکه نامگذاری تست ها، توابع، کلاس ها و متغیر های Local بکار رفته در تمام مثال ها مناسب نیستند و مانع این میشوند که بتوان از تست به عنوان Documentation استفاده کرد. با بکارگیری اسامی معنی دار میتوان این مشکل را حل کرد

الف)

این تست شرط Self-Checking بودن را رعایت نمیکند و پس از پایان تست نیاز به برسی دستی نتیجه تست داریم.

می توان با کمک توابع Assertion امکان Self-Checking را به تست مذکور اضافه کرد:

@Test

Public void testA() {

Integer result = new SomeClass().aMethod();

Assertions.assertEquals(10, result);

}

ب)

اگر نتیجه مورد انتظار در این تست، برخوردن به Exception خاصی بعد از اجرای تابع مربوطه باشد، این تست در صورت رخ دادن هر نوع خطایی fail می شود که مورد انتظار ما نیست

می توان با کمک یک روش دقیق تر Assertion این مشکل را حل کرد:

@Test

Public void testC() expects Exception {

Int badInput = 0;

Assertions.assertThrows (

ExpectedException.class,

() -> new AnotherClass().process(badInput):

);

}

ج)

در این تست برای ایجاد Fixture، از روش زنجیری استفاده شده که در آن هر تست محیط را برای تست های بعدی آماده می کند. این روش چند اشکال عمده دارد:

* ترتیب تست ها ممکن است در سیستم های مختلف تغییر کند که مانع اجرای صحیح تست ها می شود
* تست ها به هم وابسته می شوند و مطالعه هر کدام به عنوان Document به طور مستقل ممکن نیست
* موجب پیچیده شدن تست ها می شود. (اگر تستی Fail شد، مشخص نیست مشکل از خود تست است یا کد)
* جدای از این موارد بهتر است تست فرایند Initialization از بقیه تست ها جدا باشد

روش های مختلفی برای حل این مشکل میتوان پیشنهاد داد:

* حدالامکان از Fixture های Transient استفاده کنیم
* تست های درستی فرایند Initialization را از بقیه تست ها جدا کنیم
* حدالامکان به جای DOC از Double مناسب آن استفاده کنیم که نیازی به Init پیچیده نداشته باشد
* اگر Initialization فرایند پیچیده ای دارد میتوان از توابع @BeforeAll و @BeforeEach برای ساخت Fixture استفاده کرد
* بهتر است در صورتی که ساخت Fixture تاثیر ماندگار روی State برنامه دارد، بعد از پایان فرایند، این تاثیرات را به کمک توابع @AfterAll و @AfterEach از بین ببریم