گزارش کار پروژه تست 1

سید علی امام زاده 810199377، محمدحسین عقیلی 810199576

شناسه اخرین کامیت: c9388f8d4e7a327f1c950fe416a515878953503b

سوال 1)

معایب تست متد های Private:

- نقض Encapsulation: متد های Private به عنوان جزئیات پیاده‌سازی در نظر گرفته شده‌اند و تست آنها می‌تواند منجر به نقض اصل کپسوله‌سازی شود، جزئیات داخلی یک کلاس نباید مستقیماً افشا یا آزمایش شوند.

- تعمیر و نگهداری: تست متد Private می‌تواند کد را شکننده‌تر کند،چون اگر پیاده‌سازی یک روش خصوصی را تغییر دهید، ممکن است نیاز به تغییر بسیاری از تست ها را داشته باشید.

- پیچیدگی و تست غیرمستقیم: تمرکز بر تست متد های Public ، که رابط کلاس هستند، اغلب منجر به تست‌های معنادارتر و انعطاف‌پذیرتر می‌شود. تست متد Private می‌تواند مجموعه‌های تست را پیچیده کند و درک رفتار کلی یک کلاس را دشوار کند.

مزایا تست متد های Private:

- Test Coverage: تست متد های Public ممکن است Code Coverage بالاتری را فراهم کند و اطمینان حاصل کند که همه حالت های کد تست شده‌اند، که می‌تواند اطمینان صحت کد را افزایش دهد.

- ایمنی Refactoring: اگر یک متد private حیاتی است، باید آزمایش شود تا امکان Refactoring ایمن تر فراهم شود.

- کمک به اشکال‌زدایی: آزمایش متد های private می‌تواند با جداسازی و مشخص کردن مشکلات در کلاس به اشکال‌زدایی کمک کند.

اگر متد های private پیچیده ای وجود دارد یا نقش مهمی در کلاس دارند، آنها را آزمایش می کنیم. این تصمیم باید بر اساس موارد استفاده خاص و نیاز به تست کامل دارد.

Refactoring و Debugging: من ممکن است روش‌های خصوصی را هنگام فاکتورسازی مجدد کد موجود یا هنگام برخورد با سناریوهای اشکال‌زدایی چالش برانگیز آزمایش کنم.

کن بک معمولاً بر تست های نوشتن برای رابط های public تأکید می کند.

سوال 2)

Unit test یک تکنیک ارزشمند برای تأیید عملکرد تک تک اجزا کد به صورت مجزا است. با این حال، زمانی که میخواهیم کد multi-thread تست کنیم ، محدودیت‌هایی داریم.

- کد multi-thread ممکن است غیر قطعی باشد، زیرا ترتیب اجرای رشته ها می تواند بین اجراهای مختلف متفاوت باشد. Unit testing برای تولید نتایج سازگار و قابل پیش‌بینی طراحی می‌شوند که با ماهیت غیر قطعی کد multi-thread در تضاد است.
- بن بست ها می توانند در کدهای مالتی ترد زمانی رخ دهند که رشته ها مانع از پیشرفت یکدیگر شوند. شناسایی و آزمایش سناریوهای بن بست در unit test پیچیده است

با این حال برای تست کد های مالتی ترد روش هایی مانند Integration Testing و Concurrency Testing Tools وجود دارد.

به طور کلی، unit testing به تنهایی برای اطمینان از صحت کد multi thread به مشکل های بالاو پیچیدگی کافی نیست. در حالی که unit testing برای تست اجزای جداگانه ضروری هستند، باید آنها را با تست یکپارچه سازی(Integration Testing)، ابزارهای تخصصی و ... تکمیل کنیم تا به سطح بالایی از اطمینان در صحت و قابلیت اطمینان کد خود برسیم.

سوال 3)

تست1)

پیام را چاپ می کند اما هیچ ادعایی را برای بررسی صحت نتیجه انجام نمی دهد.

راه حل-

```
@Test
public void testA() {
    SomeClass someClass = new SomeClass();
    Integer result = someClass.aMethod();
    assertEquals(10, result);
}
```

تست 2)

تست انتظار استثنا دارد اما مشخص نمی کند نوع استثنا چیست. همچنین assert برای تأیید exception thrown ندارد.

راه حل-

```
@Test(expected = SomeException.class)
public void testC() {
    int badInput = 0;
    new AnotherClass().process(badInput);
}
```

تست3)

هیچ ادعایی برای تأیید موفقیت‌آمیز بودن مقداردهی اولیه ندارد. (assertion)

راه حل–

```
@Test
public void testInitialization() {
    Configuration configuration = initializeConfiguration();
    ResourceManager resourceManager = initializeResourceManager();

    assertNotNull(configuration);
    assertNotNull(resourceManager);
}
```

تست4)

Assertion به درستی نوشته نشده است.

راه حل–

```
@Test
public void testResourceAvailability() {
    boolean isResourceAvailable =
ResourceManager.isResourceAvailable("exampleResource");
    assertTrue("Resource should be available", isResourceAvailable);
}
```