Github Repository: https://github.com/PashaBarahimi/Software-Testing-Course-Projects

Last Commit: `d4bd72afeb21320076ccb189cdebffbfa03f3832`

سوال اول

در حالت کلی، نظر اکثریت بر این است که توابع private نباید تست شوند. لینک قرار داده شده در صورت پروژه نیز واضحا همین موضوع را بیان می‌کند. با توجه به اینکه توابع private از خارج یک کلاس صدا زده نمی‌شوند، توسط باقی توابع آن کلاس صدا زده می‌شوند و در واقع، توابع کمکی هستند. به همین دلیل، در اکثر موارد، می‌توانیم با تست کردن منطق توابع public، از کارکرد صحیح توابع private نیز تا حد خوبی مطمئن شویم.
اما در حالاتی ممکن است بخواهیم این توابع را نیز تست کنیم. از جمله این حالات می‌توان به موارد زیر اشاره کرد:

- اگر تابع private منطق پیچیده‌ای داشته باشد، ممکن است شامل درخت حالات بزرگی باشد و نتوان به سادگی و به کمک توابع public، منطق آن را تست کرد.
- اگر تابع private شامل یک کد legacy باشد و همچنین، فاقد API مناسب (از طریق توابع public) جهت تست باشد. در این صورت، باید تابع private به صورت مستقیم تست شود.

حال چگونه می‌توانیم این توابع را تست کنیم؟
روش تست این توابع با استفاده از نام توابع (به صورت رشته) و به کمک reflection است. پیاده‌سازی reflection در کتابخانه `java.lang.reflect` صورت گرفته است. برای مثال، کد زیر یک تابع private را از این طریق و با تغییر سطح دسترسی تابع، تست می‌کند:

```java
import org.junit.jupiter.api.Test;
import java.lang.reflect.Method;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class PrivateTest {
    @Test
    void testPrivateMethod() throws Exception {
        // Create an instance of MyClass
        MyClass myClass = new MyClass();

        // Obtain the private method using reflection
        Method privateMethod = MyClass.class.getDeclaredMethod("privateMethod",
                                                               String.class);
        privateMethod.setAccessible(true);

        // Invoke the private method on the instance
        String result = (String) privateMethod.invoke(myClass, "john");

        // Assert the result
        assertEquals("Hello john", result);
    }
}
```

سوال دوم

به کمک unit test نمی‌توانیم از صحت کارکرد یک تابع در حالت multi-threaded اطمینان حاصل کنیم. حتی اگر در تست از چند ترد استفاده کنیم، در صورتی که n بار تست اجرا شود، هیچ تضمینی نیست که در بار (n+1)-ام، fail نشود.
برای تست کردن کد multi-threaded، باید از روش‌های فرمال (برای مثال model checking) استفاده کنیم. در این روش، تمامی interleavingهای متفاوت بررسی می‌شود و نتیجه هر کدام، باید serializable باشد. از معروف‌ترین این موارد، می‌توان به JavaPathFinder اشاره کرد.

سوال سوم

1. تست اول

مشکل این تست این است که هیچ‌وقت fail نمی‌شود. در واقع ممکن است خروجی با مقداری که از آن انتظار می‌رود تفاوت داشته باشد اما خود تست pass می‌شود. همچنین، به ازای هر بار اجرای تست‌ها، باید خروجی خوانده شود تا متوجه شویم که تست pass شده یا خیر. اگر تعداد تست‌هایی که به این صورت پیاده‌سازی شده‌اند بالا باشد، انجام این کار بسیار سخت است.
در حال حاضر، معمولا پروژه‌ها شامل CI/CD هستند و هنگامی که کد به گیتهاب و یا گیت‌لب push می‌شود، به صورت خودکار تست‌ها هم اجرا می‌شوند. در آن بخش، امکان بررسی صحت تست‌ها با خواندن خروجی، کار بسیار سختی خواهد بود. اما اگر از assertEqual یا توابع مشابه استفاده کنیم، exit code این job، به راحتی نشان می‌دهد که آیا تمام تست‌ها pass شده‌اند یا خیر.

2. تست دوم

ایراد اول این است که expects یک کلیدواژه معتبر در جاوا نیست و به جای آن باید throws قرار بگیرد. با این فرض که به جای آن throws قرار دهیم، اشکال دوم این است که این تست همواره fail می‌شود. هیچ جایی ذکر نشده که از خود تست انتظار می‌رود که throw کند و این مورد، تنها اجبار جاوا است که اگر یک تابع ممکن است ارور دهد، در signature تابع ذکر شود.
روش صحیح آن این است که از تابع `assertThrows` استفاده کنیم.

3. تست سوم و چهارم

تابع تست اول، دو کلاس ذکر شده را initialize می‌کند و سپس با استفاده از assertion، درستی آن‌ها را چک می‌کند. اما، initialize کردن این توابع می‌تواند در بخش beforeEach و یا beforeAll انجام شود و در این تابع تست، فقط assertion داشته باشیم.
مشکل تست دوم جدی‌تر است. در این تست، توابع initialize دو کلاس ذکر شده اجرا نشده و در نتیجه قبل از initialization، در حال صدا زدن توابع دیگری روی آن هستیم. در حالاتی ممکن است که تابع اول زودتر از تابع دوم اجرا شود و initialization به صورت صحیح انجام شود. اما در حالت کلی، ترتیب اجرای تست‌ها ثابت نیست و ممکن است تست دوم زودتر از تست اول اجرا شود. روش اصلاح آن، یا این است که دو تابع initialize به beforeEach منتقل شوند و یا اینکه با استفاده از انوتیشن Order، ترتیب اجرای تست‌ها را مشخص کنیم.