# گزارش‌کار تمرین کامپیوتری اول آزمون نرم افزار

محمدهادی بابالو – 810199380 سنا ساری نوایی – 810199435

## سوال اول

استفاده از متدهای private در تست کار درستی نیست. برای اثبات نادرست بودن این عمل میتوان به دلایل زیر اشاره کرد:

- ما تست را برای ارزیابی رفتار کلاس از دید کاربر انجام می‌دهیم. متد های private اصولا برای پیاده‌سازی به کار می‌روند و تاثیری در دیدگاه کاربر ندارد. بطور خلاصه، در تست باید رفتار خارجی یک کلاس را تست کنیم اما تست کردن متدهای private، یعنی رفتار داخلی کلاس را تست می‌کنیم و این درست نیست.
- متدهای private باید فقط در کلاس خودشان مورد استفاده قرار بگیرند. وقتی از آن‌ها در تست استفاده کنیم، در واقع در کلاسی غیر از کلاس خودشان استفاده می‌شوند که این کار می‌تواند منجر به ایجاد مشکلاتی شود.
- همانطور که گفتیم، متدهای private برای پیاده‌سازی به کار می‌روند و اگر بخواهیم تغییری در آن‌ها ایجاد کنیم، باید تمام تست‌هایی که این متد را صدا می‌زنند را تغییر دهیم که این کار می‌تواند سخت و طولانی باشد.
- استفاده از متد private در تست خوانایی آن را کم می‌کند. یکی از کارکردهای تست این است که با استفاده از آن بتوان به منطق برنامه دست پیدا کرد و اگر خوانایی آن کم باشد، درک کد و هدف آن برای developerها سخت‌تر می‌گردد.
- وقتی در تست از متدهای private استفاده می‌کنیم، refactor کردن کد می‌تواند منجر به fail شدن برخی از تست‌های مرتبط با آن متد private شود. این اتفاق به دلیل آن است که تست‌هایمان به دلیلی استفاده از متدهای private به جزئیات پیاده‌سازی وابسته می‌شوند.

در این باره، نقل قولی معروفی از رابرت مارتین (ملقب به Uncle Bob) هم وجود دارد که می‌گوید:

*The only reason to make something private is to hide it from other classes. If you hide something, you're saying that you don't want other objects to know about it. If you don't want other objects to know about it, then you don't want to test it, because testing it would mean looking at it.*

محمدهادی بابالو – 810199380 سنا ساری نوایی – 810199435

# سوال دوم

Unit test مناسب ترین روش برای تست کد multi-threaded نیست و لزوما با استفاده از آن نمی‌توانیم از صحت این کد اطمینان حاصل کنیم. علت این است که unit test کوچکترین واحد تست است که هر جزء کوچک از کد را تست می‌کند اما کدهای multi-threaded پیچیده هستند. از دلایل دیگر میتوان به موارد زیر اشاره کرد:

- در کدهای multi-threaded ممکن است race conditions رخ دهد؛ به این معنی که نتیجه به زمان‌بندی و در هم آمیختن threadها بستگی خواهد داشت و در نتیجه هر بار ران کردن این تست‌ها ممکن است که رفتار متفاوتی رخ دهد و در نتیجه، نتیجه تست قابل اعتماد نخواهد بود.
- غیرقطعی بودن ذاتی در این کدها منجر به این می‌شود که امکان بروز رفتار اشتباه حتی در صورت پاس شدن تست‌های متناظر همچنان وجود خواهد داشت و نمی‌توان اطمینان کرد.
- امکان بروز deadlockها در کد multi-threaded وجود دارد و جلوگیری و تشخیص آن‌ها در unit testها مشکل است.
- پیچیدگی شدید ایجاد شده هم باعث می‌شود که این کار در کل پیشنهاد نشود.

محمدهادی بابالو – 810199380 سنا ساری نوایی – 810199435

# سوال سوم

- **تست اول**

این تست در کل فاقد assertion است و در نتیجه همواره pass می‌شود. در صورت استفاده از این تست در یک فرایند automated، هر بار باید به صورت دستی عبارت پرینت شده توسط method را چک کرده تا بتوانیم درستی یا نادرستی method را بفهمیم. همچنین نام‌گذاری این تست هم مناسب نیست و اطلاعاتی درباره وظیفه این تست به ما نمی‌دهد. همچنین حتی در صورت درست بودن عملکرد هم باز عبارت پرینت می‌شود که باز هم می‌تواند برای تست کردن automated مشکل‌ساز باشد.

- **تست دوم**

در این تست هم مثل تست قبل با مشکل نام‌گذاری مواجه هستیم. مشکل بعدی استفاده از expects است که یک keyword معتبر در زبان جاوا نیست. در صورتی که منظور از این keyword معادل throws باشد، مشکل این است که این تست همیشه fail می‌شود (در صورتی که badInput واقعا ورودی بدی باشد) و برای اینکه تست به درستی عمل کند باید از assertThrows استفاده کنیم. در صورتی هم که منظور از expects معادل assertThrows باشد، باز هم این مشکل وجود دارد که نوع exception مشخص نشده است و در صورت throw شدن هر exceptionای، این کیس تریگر می‌شود که باز هم راه‌حل استفاده از assertThrows است.

- **تست سوم**

در حالت عادی در JUnit هیچ تضمینی برای ترتیب اجرا شدن تست‌ها وجود ندارد و در صورت نیاز باید از annotationها استفاده کنیم تا ترتیب بین تست‌ها را مشخص کنیم. در نتیجه در اینجا ممکن است که تست دوم قبل از تست اول اجرا شود (و یا حتی به صورت تکی اجرا شود) و در نتیجه بدون اینکه initialization به درستی انجام شده باشد تست اجرا می‌شود که می‌تواند منجر به نامعتبر شدن نتیجه تست شود. همچنین در تست اول هم بهتر است که بخش اول که مربوط به initialization است به صورت متودی جدا و مثلا به صورت @BeforeEach و یا @BeforeAll وجود داشته باشد و در تست اول، صرفا assertion مناسب انجام شود. برای حل مشکل تست دوم هم می‌توانیم وابستگی بین تست‌ها را از بین ببریم و یا با annotationها ترتیب اجرای این دو تست را مشخص کنیم.

محمدهادی بابالو – 810199380 سنا ساری نوایی – 810199435

## شناسه آخرین کامیت

194b19b2027a88c52990723526e3939564ee61dc

https://github.com/Hadi-loo/Software-Testing-Course