# به نام او

شهنام فیضیان 810100197
عرفان دارایی 810100139

آزمون نرم‌افزار
گزارش بخش تئوری پروژه 1

آدرس مخزن https://github.com/ShahnamFeyzian/Software-Testing-Course

شناسه آخرین کامیت 0b3bbe670607d238189d2692d59f0b8ecbfcd6f4

---

## سوال اول

کلاس Assume در JUnit متد های assumption ای را فراهم می‌آورند که در صورت برقرار نبودن شرط آنها تست ignore می شود. از جمله این متدها می‌توان به assumeTrue, assumeFalse, assumeNotNull و assumeThat اشاره کرد. متدهای assumption در مواقعی کاربرد دارند که می‌خواهیم تست فقط در شرایط خاصی اجرا شود، برای مثال پیش‌نیازهای اجرای تست فراهم باشد یا محیطی که تست روی آن انجام می‌شود حتما یک سری ویژگی‌های مشخص را دارا باشد. عموما وقتی از assumption ها استفاده می‌کنیم داریم این مفهوم را منتقل می‌کنیم که اجرای این تست بدون برقراری شرایط assumption معنا ندارد، بنابراین علاوه بر قابلیت‌هایی که به ما می‌دهد ابزاری مناسب برای رسیدن به هدف test as documentation & specification است. البته اگر به خوبی و در مواقع درست از آن استفاده شود. در زیر چند نمونه از سناریوهایی که assumeTrue در آن کاربرد دارد آورده شده است.

- اجرای تست فقط روی یک سیستم عامل خاص. (البته راه‌های بهتری مثل استفاده از annotation های مخصوص این کار نیز وجود دارد)
- رد کردن تست، اگر ارتباطمان با دیتابیس قطع است.

- اجرای تست فقط در شرایطی که یک متغیر محیطی (environment variable) خاص ست شده باشد.
- رد کردن یا اجرای تست به ازای یک ورژن خاصی از جاوا. (مانند مورد یک annotation نیز دارد)
- در مواقعی که ورودی تست‌هایمان به صورت رندوم ساخته می‌شوند و ...

## سوال دوم

خیر، یکی از مهم‌ترین پیچیدگی‌هایی که کدهای multi-threaded آن را ایجاد می‌کند عدم قطعیت گرفتن جواب یکسان به ازای ورودی یکسان است. این امر به  این دلیل رخ می‌دهد که ترتیب اجرای بخش‌هایی که به صورت همروند اجرا می‌شوند نامشخص و عملا غیر قابل پیش‌بینی است. پس نمی‌توان برای اطمینان از درست کار کردن یک کد multi-threaded به unit testing اکتفا کرد و باید از روش‌های فرمال و یا ابزارهای مخصوص این کار استفاده کرد.

## سوال سوم

برای تشخیص تست‌هایی که fail شده‌اند باید نتیجه تک‌تک تست‌ها را از روی کنسول به صورت manual بخوانیم که هزینه زمانی بسیار زیادی دارد و تکرار کردن مکرر تست‌ها را مختل می کند که همین اجرا نشدن مکرر تست‌ها تبعات فراوانی در پی دارد.
همچنین امکان خطا در پیدا کردن همه تست‌هایی که fail شده‌اند زیاد است و ممکن است تستی که اجرای موفقی نداشته از زیر دستمان در برود.

# سوال چهارم

## الف)

بجای استفاده از expects باید از assertThrows استفاده کنیم تا هم تست در مواقعی که مشکلی وجود ندارد fail نشود و باعث اختلال در pipeline و مراحل automation نشود. و هم خوانایی بیشتری داشته باشد.

## ب)

مشکل اصلی این تست‌ها این است که fixture بین آنها share است، به عبارتی اجرای این تست‌ها روی استیت سیستم تغییراتی را اعمال می‌کند که در آخر تست آن اثرات را پاک نمی‌کنند و تست بعدی روی استیتی که تست قبلی آن را تغییر داده است اجرا می‌شود. و از آنجایی که این امر باعث ایجاد وابستگی بین تست‌ها می شود کار درستی نیست. تست‌های واحد باید خودشان در اول تست fixture خود را بسازند و در آخر نیز اثراتشان را پاک کنند. که به منظور انجام ساده تر این کار می توانند از before each, before all, after each و after all استفاده کنند.