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

Last Commit: `68bbca645fe2552aec276766c710fd8ca5f1b836`

سوال اول

گاهی برای استفاده از وابستگی‌های یک کلاس، کلاس مد نظر را داخل خود می‌سازیم. با این کار coupling کلاس ما به آن زیادتر شده و در صورت تغییر آن، کد کلاس هم باید تغییر داد.
یک الگو برای حل این مشکل، Dependency Injection است که از اصول SOLID نیز می‌باشد. در این الگو، وابستگی‌های خارجی به کلاس تزریق شده و از ساخت و استفاده مستقیم آنها در کلاس جلوگیری می‌شود.
در تست کردن، از این الگو جهت کار با test double-ها استفاده می‌شود. با استفاده از این الگو کنترل ما روی تست بیشتر می‌شود؛ چون که وابستگی، پارامتری از کلاس شده و در داخل آن پنهان نیست. این یعنی می‌توان به راحتی از بیرون آنها را تغییر داد و مثلا یک mock یا هر test double دیگری را جایگزین آن کرد.
برای پیاده‌سازی این الگو، معمولا از سه راه استفاده می‌کنند.

- Constructor Injection: در این روش، وابستگی‌های خارجی در کانستراکتور کلاس مدنظر داده می‌شوند. یعنی سازنده اینستنس کلاس موظف است وابستگی‌ها را به کلاس بدهد. از آنجا که وابستگی از پارامترهای کانستراکتور شده است، از بیرون مشخص بوده و جهت ساخت آبجکت حتما داده می‌شود. در این حالت قبل از ساخت کلاس مطمئن هستیم که وابستگی‌ها وجود دارند.
- Setter Injection: در این روش، وابستگی‌های خارجی (معمولا به صورت فیلدهای پرایوت) از طریق setter-های داخل کلاس مقداردهی می‌شوند. این روش انعطاف‌پذیری بیشتری داشته و می‌توان در حین وجود آبجکت، وابستگی‌ها را تغییر داد یا وابستگی‌های اختیاری داشت. با این روش اطمینان کامل از اینکه همه وابستگی‌های کلاس در هنگام استفاده از آن وجود دارند و set شده‌اند نداریم.
- Field Injection: در این روش، وابستگی‌های خارجی مستقیم به فیلدهای کلاس تزریق می‌شوند و از متدها استفاده نمی‌شود.

استفاده از این روش‌ها بستگی به شرایط دارد و در حالت کلی، برای dependency-های اجباری کلاس، استفاده از Constructor Injection راه بهتری است و در حالات دیگر می‌توان از Setter Injection استفاده کرد. روش Field Injection کمتر در تست استفاده می‌شود.

سوال دوم

.1

test double-ها برای جعل و وانمود کردن به جای آبجکت‌های واقعی استفاده می‌شوند. استفاده از کلمه imposter این معنی را می‌رساند که test double-ها آبجکت‌های اصلی نیستند و جایگزینی جهت تست کردن ایزوله و کنترل شده اند.

.2

جهت جایگزینی وابستگی‌های خارجی و تست ایزوله و کنترل‌شده، از test double-ها استفاده می‌شود که به انواع زیر تقسیم می‌شوند:

- Dummy Object: این نوع test double، زمانی استفاده می‌شود که متدی به پارامتری نیاز دارد که تأثیری بر سناریو تست‌مان ندارد. در این صورت از Dummy Object-ها به عنوان placeholder استفاده می‌شود.
- Test Stub: این نوع test double، به گونه‌ای است که به ازای فراخوانی‌های متدهای مختلف، جواب و داده‌ای از پیش تعیین شده را ریترن می‌کند. stub-ها به رفتارهایی که برای آن تعیین نشده پاسخی نمی‌دهد و برای شبیه‌سازی رفتاری خاص از وابستگی مدنظر استفاده می‌شود. با این کار، تعاملات با وابستگی‌های خارجی کاملا طبق انتظار قبل بوده و حالت غیر قابل پیش‌بینی رخ نمی‌دهد که در ایزوله کردن تست کمک می‌کند.
- Mock Object: این نوع test double، کمی شبیه stub است ولی stub برای state testing استفاده می‌شود درحالی که mock برای behavior verification استفاده می‌شود. این یعنی stub صرفا جواب تعیین شده‌ای به متدی که از آن صدا می‌شود می‌دهد ولی برای mock-ها انتظاراتی تعیین می‌شود که چطوری در حین تست استفاده می‌شوند. مثلا می‌توان ترتیب و تعداد صدا شدن متدها، آرگومان‌های داده شده به متدها و چیزهای دیگر را بررسی کرد. mock-ها می‌توانند کمی منطق داشته باشند و مناسب با سناریو تست مقادیر متفاوتی ریترن کنند.
- Test Spy: این نوع test double، رفتار واقعی آبجکت اصلی را نگه می‌دارد و این اجازه را می‌دهد که درباره متدهای صدا شده و تعاملات آن اطلاعاتی دریافت کنیم و برخی رفتارها را تغییر دهیم. اینها به نوعی partial mock هستند که یعنی فقط بخشی از رفتار آن تغییر کرده و بقیه آن مانند خود آبجکت واقعی است.
- Fake Object: این نوع test double، یک پیاده‌سازی ساده از کلاس مدنظر است. از fake-ها می‌توان برای جایگزینی منطق پیچیده برخی کلاس‌ها استفاده کرد و آنها را با منطقی ساده‌تر و سبک‌تر جایگزین کرد. در اینجا یک تعادل بین ساده بودن و طبیعی بودن رفتار کلاس برقرار می‌شود. یک مثال مشهور از این دسته test double-ها، جایگزینی دیتابیس واقعی با دیتابیسی in-memory است که موجب افزایش سرعت تست می‌شود.

سوال سوم

می‌توان گفت تفاوت اساسی بین تست کردن classical و mockist این است که mockist به behavior verification و interaction-ها اهمیت می‌دهد (تعامل‌های unit و وابستگی‌ها) درحالی که classical به state نهایی سیستم تست توجه می‌کند.

در mockist از mock-ها به عنوان test double به مقدار زیاد استفاده می‌شود تا رفتار و انتظارات بررسی شوند. در classical کمتر از mock-ها و بیشتر از خود آبجکت‌های اصلی استفاده می‌شود. این یعنی در mockist ستاپ بیشتری قبل از تست داریم و وقت بیشتری برای نوشتن می‌گیرند.

assertion-ها در mockist معمولا بیشتر بوده و در classical بیشتر به نتیجه نهایی توجه می‌شود و نه مراحل میانی. تست‌ها در classical در حد mockist ایزوله نیستند چون که با آبجکت‌های واقعی کار می‌کنند و وابستگی وجود دارد.

تست classical برای integration و system testing بیشتر به کار میاید و mockist اهداف test driven development داشته و برای unit test می‌تواند بهتر باشد.

مزایای classical:

- نوشتن و پیاده‌سازی تست‌ها راحت‌تر است. تست‌ها در کل فهم ساده‌تری دارند.
- این تست‌ها از آبجکت‌های واقعی استفاده می‌کنند و به این خاطر طبیعی‌تر و پایدارتر اند و اطمینان بیشتری از درستی کلیت سیستم می‌دهند.
- سربار ستاپ کردن mock-ها در اینجا وجود ندارد و اجرای تست‌ها سریع‌تر خواهد بود.
- از آنجا که از اصل آبجکت‌ها استفاده می‌شود، نیاز به maintain کردن mock-ها در صورت تغییر منطق یک وابستگی خارجی نیست.

معایب classical:

- به دلیل چک نشدن مراحل میانی توابع و تمرکز بر حالت نهایی، ممکن است برخی ایرادها نادیده گرفته شوند.
- گاهی سرویس‌هایی نیازمند شبیه‌سازی اند و استفاده و ستاپ نسخه واقعی آنها پرهزینه خواهد بود و بهتر است که از mock-ها استفاده شود.
- یافتن دلیل fail شدن تست در این روش سخت‌تر است چون تعاملات زیادی در یک تابع رخ می‌دهد و این روش فقط نهایت کار را بررسی می‌کند.

مزایای mockist:

- رفتار سیستم طبق انتظارات بررسی می‌شود که می‌تواند مشکلات بیشتری نسبت به تست state نهایی را تشخیص بدهد.
- دلیل fail شدن تست‌ها به طور دقیقی مشخص خواهد بود چون که assert-ها در بخش‌های مختلف تابع استفاده می‌شوند.
- در این روش به دلیل mock کردن وابستگی‌ها، تست‌ها ایزوله خواهند بود و مطمئنیم که مشکل از خود unit مد نظر است.
- استفاده از mock-ها سربار ستاپ کردن دارد ولی ممکن است در حالاتی از آبجکت اصلی سریع‌تر بشود.

معایب mockist:

- سربار ستاپ کردن mock-ها و گاهی پیچیدگی آنها.
- در صورت زیاد بودن وابستگی‌ها تست کردن سخت می‌شود. در حالت کلی نوشتن تست در این روش زمان بیشتری می‌برد.
- در صورت تغییر منطق یک وابستگی خارجی، منطق mock نیز باید تغییر کند که یعنی maintain کردن این روش کار بیشتری دارد.
- mock-ها ممکن است خیلی طبیعی نباشند و شرایط واقعی اجرای متد را شبیه‌سازی نمی‌کنند.