پروژه اول درس تست

آرین حدادی 810196448

امیر پورمحمدعلی 810196435

**سوال 3)**

در رابطه با تست کردن متد های private می توان گفت که یونیت تست متد های private در کل توصیه نمی شود و در سایت <http://shoulditestprivatemethods.com> هم که در صورت پروژه به آن اشاره شده است(که Kent Beck آن را توییت کرده است) هم صراحتا در جواب سوال Should I test private methods گفته است نه.

در حالت کلی هم نیازی به تست آن ها نخواهد بود زیرا میتوان با تست متد های پابلیک که خودشان برای پیاده سازی منطق خود متد های private را صدا میزنند، از عملکرد درست آن متد های private نیز اطمینان حاصل کرد.

یونیت تست در واقع مانند مشتری است.مشتری با متد های privateکاری ندارد بلکه از طریق رابط های public به برنامه دسترسی دارد و لذا همان ها هم تست می شوند.و اگر تست کردن یک بخشی از منطق برنامه از طریق رابط های public سخت است احتمالا هنگام عرضه نرم افزار هم استفاده از آن سخت خواهد بود و این نشان می دهد که نیاز است در طراحی نرم افزار بازبینی داشته باشیم.همچنین اگر تست کردن متدی public به قصد تست کردن متد های private به گونه ای باشد که متد public متد های private زیادی را صدا می زند و دقیق نمیتوانیم متد های private را یکی یکی تست کنیم و این باعث شود که بخواهیم متد های پرایوت را جداگانه تست کنیم، این حالت هم می تواند نشان دهنده این باشد که متد های ما از یکی از قوانین clean coding که بیان می کند یک متد نباید تعدادی زیادی responsibility داشته باشد و باید یک کار انجام دهد را در طراحی کد نقض کرده ایم و نیاز به اصلاحاتی در نحوه نوشتن کد داریم.

و لذا در حالت کلی تست کردن متد های پرایوت توصیه نمی شود و نیازی هم به آن ها نیست.اما اگر برخلاف توصیه ها کسی بخواهد آن را تست کند یا باید آن را پابلیک کند یا از ریفلکشن استفاده کند.

ما در حین زدن پروژه در متد getVisitsInternal() به یک متد private برخوردیم اما آن را تست نکردیم.زیرا از این متد در متد public به نام getVisits() استفاده می شود و زمانی که این متد تست شده و درست عمل می کند لذا میتوان اطمینان داشت که این متد private هم به طور صحیح کار می کند که این حالت همان حالتی است که بالاتر هم ذکر شد که از طریق تست متد های public به صحت عملکرد متد های private پی ببریم.

**سوال 4)**

تست کردن در threading بسیار مشکل است. تقریبا نمی توان با test اثبات کرد که یک کد thread-safe است. زیرا یک کد که unsafe است ممکن است میلیون ها بار بدون مشکل شود اجرا شود و بار بعد به error بخورد. زیرا همه چی به ترتیب اجرای process ها در آن لحظه دارد و روی هر سیستم و هر بار ران ممکن است این ترتیب متفاوت باشد و تنها در یک حالت به مشکل بخورد.

تنها راه اثبات thread-safe بودن یک کد داشتن دانش کافی درباره آن زبان و static analysis کردن آن کد می باشد. پس unit test کردن برای یک کد به صورت multi-thread تقریبا هیچ فایده ای ندارد. می توان مثلا برای نشان دادن error های مربوط به threading در یک کد، اگر سیستم کشش n تا thread را دارد، آن را با 2n تا thread پر کرد تا لود آن زیاد شود و احتمال بروز آن ارور ها زیاد تر شود ولی در حالت کلی هیچ تضمینی برای بروز ارور در این حالت نیست و برخی ارور ها ممکن است در لود سبک یا متوسط خود را نشان دهند.

پس unit-test ها آن طور که در کد های معمولی در به اطمینان رسیدن ما نسبت به کد کمک می کنند در multi-thread ها هیچ تضمینی به ما نمی دهند زیرا با اجرای اینجور کد ها در platform های مختلف تنظیمات آن ها را به طور دقیق از جمله تعداد process ها، هسته ها و cache های آن را نمی دانیم. برای تست آن ها باید ابتدا مطمئن شویم در خود unit ها هیچ مشکلی نیست و تنها مشکل از concurrency می باشد و سپس می توان با ایجاد لود بالا و شرایط برای unit ها احتمال رخداد ارور ها را بیشتر کنیم.

در کلاس Pet به عنوان مثال می توان مثال هایی زد که unsafe بودن آن را دید. به عنوان مثال دو thread داریم t1 و t2 که هر دو دارند به یک object از Pet دسترسی پیدا می کنند. به عنوان مثال این pet دارای birthdate ای در سال 2000 می باشد. مثلا t1 دارد با استفاده از با متد getVisitsUntilAge تمام visit های کمتر از 5 سال را می گیرد. در این میان که این t1 دارد روی visit ها iterate می کند به عنوان مثال t2 می آید و birthdate را به سال 2003 تغییر می دهد. پس t1 از این به بعد دیگر visit هایی که با فرض 2000 بودن birthdate داشته می گرفته را نمی گیرد و از اینجا به بعد با 2003 را می گیرد. پس در اینجا inconsistency داریم و unsafe است. حال هیچ unit test ای نمی تواند این شرایط را به دقت تست کند و ممکن است بار ها تست اجرا شود و به درستی t2 بعد از کار t1 اجرا شود ولی ممکن هم است که t2 یک بار در وسط کار t1 اجرا شود و تست ما چون نمی داند کدام process روی platform ما زودتر اجرا می شود نمی تواند به درستی این حالت را تست کند.

سوال 5)

تست اول برای چک کردن نیاز به چک دستی و مشاهده خروجی برنامه دارد که ایراد دارد.زیرا همانطور که در درس گفته شده است بهتر است تست ها اتوماتیک باشند و نیازی به چک کردن دستی نباشد زیرا به مراتب سریعتر و راحت تر بوده و روند تست بسیار بهتر و راحت تر انجام می شود.

تست دوم علاوه بر اینکه اتوماتیک نیست و ایراد اتوماتیک نبودن را دارد بلکه اصلا خروجی را در جایی ذخیره نمیکند که بخواهد همان دستی هم چک شود همچنین چون متد bar() که تست می شود ممکن است Exception ای throw کند و ما بلاک try catch نداریم ممکن است به مشکل دومی بخوریم که آن هم قطع شدن روند تست بخاطر رخ دادن Exception است.

تست سوم ولی چون هدفش exception خوردن است(چون نوشته شده expects Exception) و ورودی ای که داده می شود هم badInput نام دارد(یعنی از نامش پیداست که ورودی بدی است) لذا تست درواقع اتوماتیک محسوب میشود زیرا هدف رخ دادن Exception بوده و اگر رخ ندهد تست fail میشود پس تست، اتوماتیک محسوب می شود و بر خلاف دو تست قبلی مشکلی ندارد.

**باگ های یافت شده)**

**یکی از باگ های یافت شده این است که اگر ورودی متد addVisit را null بدهیم با اینکه Exception ای ایجاد میشود اما به تعداد اعضای فیلد visits افزوده می شود و لذا متد addVisit() را به گونه ای تغییر دادیم که درصورت null بودن چیزی اضافه نکند و یک Exception را throw کند.**