به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر

آزمون نرم افزار

گزارش کار شماره 1

محمد پویا افشاري (810198577)
مصطفی ابراهیمی (810199575)

بخش دوم - گزارش کار

1. سوال اول:
   - معمولا Private method ها، به این علت استفاده می‌شوند که جزییات آنها در دسترس کاربر ها قرار نگیرد این در حالي است که نوشتن Test به صورت Documentation میتواند اصل Encapsulation این جزییات را بشکند. در راه حل تحصين برانگيز ارايه شده توسط jop ، به این نکته اشاره مي‌شود که در صورتي که تست هاي توابع خصوصي ما به اندازه اي مهم و بزرگ باشند که نیاز حتمي به تست کردن دارند میتوان آن را به کلاس تبدیل کرده و در آنجا به صورت Public براي متد خواسته شده ی قبلی تست نوشت. نمونه انجام این کار در Method Object موجود است. (البته در ایراد همین روش میتوان به ایجاد Component های بدون استفاده در برنامه در صورت کاربرد بیش از حد اشاره نمود.)
   - همچنین نقطه نظرهای بیشتری در زمینه تست متدهای Private میتوان در نظر گرفت. در مرحله Refactor کدها معمولا متدهای Private بیشترین تغییر را می‌کنند.وهمین امر باعث شکننده بودن تست های نوشته شده و اضافه کردن Overhead تست می‌شود.
   - برخی از زبان ها مثل جاوا به راحتی امکان تست مستقیم متدهای Private را نمی‌دهند به همین منظور در صورت استفاده از Reflection و ابزارهای مشابه برای گرفتن دسترسی عملا Bad practiceانجام میدهیم.

2. سوال دوم:
   - تست کدهای Multi-threaded دارای دشواری هایی فراوانی است به طور مثال:
     - ترتیب اجرا: در هربار اجرا برنامه thread ها با ترتیب متفاوتی اجرا می شوند و هربار با یک روند اجرا برنامه روبرو میشویم.
     - در برخی سناریوها ممکن است چند thread همزمان وارد رقابت بر سر منابعی مشترکی شوند و میتواند منجبر به deadlock میشود.
     - ممکن است بین بعضی thread ها به نتایج برخی دیگر از thread ها وابسته باشند. و برخی از ترتیب اجرا ها باعث بیکار شدن thread ها در انتظار برخی دیگر از thread ها شود.

- به علت وجود عدم قطعیت در نحوه روند اجرا کد های Multi-threaded، در تست آنها با دشواری هایی همراه هستیم. به همین منظور پیشنهاد میشود که این قبیل کد ها چندین بار ران شوند و در حین ران شدن log گرفته شود زیرا احتمال دارد صرفا برخی از ترتیب ران شدن thread ها منجر به اجرا نادرست کد شود و باید آن ترتیب را به وسیله log پیدا کنیم.
- در این زمینه ابزارهایی برای ساده تر شدن فرایند تست به وجود آمده اند که میتوان به **Java Concurrency Stress (jcstress)** اشاره کرد.

3. سوال سوم:

```
@Test
public void testA() {
    Integer result = new SomeClass().aMethod();
    System.out.println("Expected result is 10. Actual result is " + result);
}
```

در تست داده شده خروجي گرفته شده از متد چاپ مي‌شود در اين حالت هيچ assertion وجود ندارد كه درستي اين برابري را چك بكند. به جاي اين كار ميتوانستيم مقدار خواسته شده مورد انتظار 10 را در متغير expected نگه داري كنيم و برابري را با عمل Assertions.assertEquals(expected, result) بررسي كنيم.

```
@Test
public void testC() expects Exception {
    int badInput = 0;
    new AnotherClass().process(badInput);
}
```

تست داده شده در اين حالت assertion خاصي ندارد براي رفع اين مشكل كه در صورتي كه ورودي بدي بدهد خروجي يك Exception درست Throwبكند بايد از assertThrows استفاده كنيم. براي اين كار به جاي خط new ميتوانيم از

```
Assertions.assertThrows(Exception.class, () -> {
    new AnotherClass().process(badInput);
});
```

استفاده بكنيم.

```
@Test
public void testInitialization() {
    // Initialize the configuration and resources
    Configuration.initialize();
    ResourceManager.initialize();
    // Perform assertions to validate the initialization
    // ...
}

@Test
public void testResourceAvailability() {
    // Check if a specific resource is available
    boolean isResourceAvailable = ResourceManager.isResourceAvailable("exampleResource");
    assertTrue(isResourceAvailable);
}
```

در اين مورد testInitialization اين طور به نظر ميرسد كه محيطي را setup مي‌كند كه تست resourseAvailability از آن استفاده مي‌كند كه اين امر مصداق chained بودن مي‌باشد. به جاي اين كار ميتوان از beforeEach استفاده كرد.