آزمون نرم‌افزار تمرین اول

بخش تئوری

امیر پارسا موبد | ۸۱۰۱۰۰۲۷۱

ارشیا عطایی | ۸۱۰۱۰۰۲۵۲

آدرس مخزن : https://github.com/ParsaMOBB/SWT-Course-Projects

شناسه آخرین کامیت : 97b52ef791bb53d84ba3325fc4f8931a8fd65ff2

سوال ۱

در زبان جاوا، متدهای Assume از کتابخانه‌های تست (مانند JUnit) برای کنترل اجرای تست‌ها استفاده می‌شوند. هدف این متدها این است که مشخص کنند آیا شرایطی برای اجرای یک تست مهیا است یا نه. اگر شرط برقرار نباشد، تست به جای شکست خوردن، نادیده گرفته می‌شود.

متد AssumeTrue:

این متد برای بررسی یک شرط بولی به کار می‌رود. اگر شرط true باشد، تست اجرا می‌شود، و اگر false باشد، تست نادیده گرفته می‌شود (نه اینکه شکست بخورد). این روش برای تست‌هایی که وابسته به محیط، پیکربندی، یا شرایط خاصی هستند، مناسب است.

مثال:

```
@Test
public void testOnlyOnWindows() {

Assume.assumeTrue(System.getProperty("os.name").startsWith("Windows"));
}
```

در این مثال، اگر سیستم‌عامل ویندوز نباشد، تست نادیده گرفته می‌شود.

سوال ۲

می‌توان از آزمون واحد برای تست کدهای چند-نخی استفاده کرد، اما این روش به دلیل وجود مشکلات همزمانی و شرایط رقابتی (race conditions) به سادگی نمی‌تواند تمامی جوانب ایمنی رشته‌ها (thread-safety) را تضمین کند. یکی از چالش‌های اصلی این است که مشکلات همزمانی معمولاً تصادفی و غیرقابل پیش‌بینی هستند، بنابراین در یک اجرای واحد از تست ممکن است ظاهر نشوند. همچنین، بسیاری از این مشکلات تحت شرایط زمانی خاصی رخ می‌دهند که ممکن است در طول یک تست کوتاه دیده نشوند.

برای اطمینان بیشتر، بهتر است از ابزارهای تخصصی همزمانی مثل **JCStress** استفاده شود که توانایی شبیه‌سازی شرایط پیچیده‌تر را دارند. همچنین، استفاده از سازوکارهای مناسب همزمانی، مانند Lock، synchronized یا کلاس‌های java.util.concurrent، از ابتدا ضروری است. در نهایت، آزمون واحد به‌عنوان بخشی از یک رویکرد جامع می‌تواند مفید باشد، اما به‌تنهایی نمی‌تواند thread-safety را تضمین کند.

سوال ۳

استفاده از کنسول برای پرینت خروجی به‌جای Assert چند ایراد دارد:

1. **بررسی دستی نتایج**: با پرینت به کنسول، باید نتایج را دستی بررسی کنید، در حالی که Assert بررسی خودکار دارد.
2. **گزارش‌دهی نامناسب**: Assert در صورت شکست، گزارش دقیق و traceback ارائه می‌دهد، اما پرینت این ویژگی را ندارد.
3. **عدم مقیاس‌پذیری**: بررسی دستی در پروژه‌های بزرگ زمان‌بر است، ولی Assert به‌صورت خودکار نتایج را مدیریت می‌کند.
4. **عدم سازگاری با CI/CD**: ابزارهای CI نیاز به گزارش خودکار دارند، که با Assert فراهم می‌شود، اما پرینت چنین قابلیتی ندارد.

سوال ۴

الف) مشکل آزمون:

1. **نحوه انتظار استثنا**: عبارت expects Exception اشتباه است.
2. **نوع استثنا**: استفاده از Exception عمومی دقیق نیست؛ بهتر است نوع خاص استثنا مشخص شود.

راه‌حل: از assertThrows در Junit 5 یا @Test(expected = SomeException.class) در Junit 4 استفاده کنید:

```
@Test
public void testB() {
    int badInput = 0;
    assertThrows(IllegalArgumentException.class, () -> new
AnotherClass().process(badInput));
}
```

ب)

مشکل این کد در وابستگی بین تست‌هاست، زیرا هر دو تست از یک نمونه مشترک Calculator استفاده می‌کنند. این می‌تواند منجر به نتایج غیرمنتظره شود.

مشکلات:

1. **وابستگی بین تست‌ها**: تغییر حالت Calculator در یک تست می‌تواند بر تست دیگر تأثیر بگذارد.

راه‌حل:

1. **ایجاد یک نمونه جدید برای هر تست**: استفاده از @BeforeEach برای بازنشانی حالت Calculator قبل از هر تست.

مثال:

```
public class TestCalculator {
    Calculator fixture;

    @BeforeEach
    public void setUp() {
        fixture = Calculator.getInstance();
        fixture.setInitialValue(0);
    }
```

```
    @Test
    public void testAccumulate() {
        int result = fixture.accumulate(50);
        Assertions.assertEquals(50, result);
    }

    @Test
    public void testSubsequentAccumulate() {
        int result = fixture.accumulate(100);
        Assertions.assertEquals(100, result);
    }
}
```