آزمون نرم‌افزار

تمرین تئوری سری ۲

حوریه سلطانی-۴۰۰۱۹۱۷۳

آ)

تست خودکار به استفاده از نرم‌افزار برای اجرای خودکار تست‌ها، مقایسه نتایج واقعی با نتایج مورد انتظار، آماده‌سازی شرایط اولیه تست، کنترل فرآیند تست و گزارش‌گیری از نتایج گفته می‌شود. این کار به جای اجرای دستی تست‌ها توسط انسان، با استفاده از کد و ابزارهای خاص انجام می‌گیرد.

سه مزیت اصلی تست خودکار:

1. کاهش هزینه (Cost Reduction):

با تست خودکار، تست‌ها می‌توانند بارها بدون هزینه اضافی اجرا شوند. نیازی به نیروی انسانی برای اجرای مجدد تست‌ها نیست، بنابراین در پروژه‌های بزرگ یا تکرار شونده، هزینه‌ها به شکل چشمگیری کاهش می‌یابد.

2. کاهش خطای انسانی (Reduced Human Error):

اجرای تست‌ها به صورت دستی ممکن است با اشتباهاتی همراه باشد؛ مانند وارد کردن داده‌ی نادرست یا اشتباه در بررسی نتایج. اما تست خودکار با اجرای دقیق و یکسان در هر بار اجرا، احتمال خطا را به حداقل می‌رساند.

3. کاهش چشمگیر هزینه تست رگرسیون (Regression Testing):

در پروژه‌هایی که به طور مرتب تغییر می‌کنند، تست‌های قبلی باید بارها اجرا شوند. تست خودکار این فرآیند را بسیار سریع‌تر و ارزان‌تر انجام می‌دهد، بنابراین هزینه و زمان تست‌های رگرسیونی کاهش می‌یابد.

ب)

یک تست کیس (Test Case) یک ابزار چندبخشی است که دارای ساختار مشخصی می‌باشد. دو جزء اصلی آن عبارت‌اند از:

۱. مقادیر ورودی (Test Case Values)

مقادیر مورد نیاز برای اجرای نرم‌افزار تحت تست. این مقادیر مشخص می‌کنند که هنگام اجرای تست، چه داده‌هایی به برنامه داده می‌شود. برای مثال یک متد داریم که تعداد اعداد صفر در آرایه را می‌شمارد:

```
int countZero(int[] arr);
```

مقدار ورودی می‌تواند: `[0, 1, 2, 0]` باشد.

۲. نتیجه مورد انتظار (Expected Results)

نتیجه‌ای که در صورت رفتار صحیح نرم‌افزار باید تولید شود. این مقدار به ما می‌گوید که اگر برنامه درست کار کند، خروجی باید چه باشد. تست موفق است اگر نتیجه واقعی با این مقدار برابر باشد. بر اساس مثال قبل برای ورودی `[0, 1, 2, 0]`، نتیجه مورد انتظار باید `2` باشد چون دو عدد صفر داریم.

نقش تست اوراکل (Test Oracle)

تست اوراکل ابزاری است برای مقایسه نتیجه واقعی با نتیجه مورد انتظار، و مشخص کردن اینکه تست قبول شده یا شکست خورده. در جاوا معمولاً این کار با متدهایی مثل `assertEquals()` در JUnit انجام می‌شود.

مثال:

```
assertEquals(2, countZero(new int[]{0, 1, 2, 0}));
```

اگر مقدار واقعی برابر ۲ باشد، تست پاس می‌شود. در غیر این صورت، شکست می‌خورد و یعنی برنامه نیاز به اصلاح دارد.

ج)

JUnit یک فریم‌ورک متن‌باز برای زبان Java است که به منظور نوشتن و اجرای تست‌های واحد (Unit Tests) طراحی شده است. این فریم‌ورک امکان اجرای خودکار تست‌ها، گزارش‌دهی نتایج و بررسی صحت عملکرد کد را فراهم می‌سازد.

سه ویژگی مهم JUnit:

1. **استفاده از Assertions** برای بررسی خروجی‌ها و مقایسه نتایج واقعی با نتایج مورد انتظار.
2. **پشتیبانی از Fixtures** برای اشتراک‌گذاری و مدیریت وضعیت تست‌ها قبل و بعد از اجرا.
3. **پشتیبانی از Test Suites و Test Runners** برای سازمان‌دهی، اجرای گروهی تست‌ها، و ارائه گزارش‌های متنی یا گرافیکی.

JUnit قابلیت استفاده در خط فرمان یا محیط‌هایی مانند Eclipse را نیز دارد

د)

- **تست متد (Test Method):** هر متد مستقل که رفتار بخشی از نرم‌افزار (معمولاً یک تابع) را بررسی می‌کند.
- **تست کلاس (Test Class):** مجموعه‌ای از تست متدها به همراه کدهای آماده‌سازی و پاک‌سازی قبل و بعد از هر تست.

اجزای تست کلاس:

1. یک یا چند متد تست (`@Test`)
2. متد آماده‌سازی با `@Before` برای مقداردهی اولیه (fixture)
3. متد پاک‌سازی با `@After` برای آزادسازی منابع
4. تعریف متغیرهای سراسری برای اشتراک در تست‌ها

ه)

Fixture به مجموعه‌ای از **متغیرها و اشیاء مشترک** بین تست‌ها گفته می‌شود که وضعیت اولیه تست را تعریف می‌کنند.

- متد حاوی `@Before`: قبل از هر تست اجرا می‌شود و وظیفه آماده‌سازی fixture را دارد.
- متد حاوی `@After`: پس از هر تست اجرا می‌شود و وظیفه پاک‌سازی fixture را دارد.

مثال:

```java
private List<String> list;

@Before
public void setUp() {
    list = new ArrayList<>();
}

@After
public void tearDown() {
    list = null;
}
```

و)

از **Assertions** در **JUnit** استفاده می‌شود تا صحت خروجی‌ها را بررسی کنیم. این متدها بررسی می‌کنند که نتیجه واقعی با مقدار مورد انتظار مطابقت دارد یا نه. اگر مغایرتی باشد، تست شکست می‌خورد و به توسعه‌دهنده اطلاع داده می‌شود.

چند متد اصلی:

`assertEquals(expected, actual)`-
بررسی می‌کند که مقدار واقعی با مقدار مورد انتظار برابر است.

`assertTrue(condition)`-
بررسی می‌کند که شرط داده‌شده درست است.

`assertFalse(condition)`-
بررسی می‌کند که شرط داده‌شده نادرست است.

`assertNull(object)`-
بررسی می‌کند که شیء موردنظر `null` است.

`assertNotNull(object)`-
بررسی می‌کند که شیء موردنظر `null` نیست.

این متدها در تست متدها استفاده می‌شوند و در صورت نقض شرط، گزارش شکست تولید می‌کنند.

ز)

در اینجا مثالی از تست پارامتریک با استفاده از `@RunWith(Parameterized.class)` و `@Parameters` ارائه می‌شود که این شرایط را برآورده می‌کند:

توضیح ساختار کد:

- از `@RunWith(Parameterized.class)` برای اجرای پارامتریک استفاده شده است.
- در متد `data()`، لیستی از آرایه‌های آبجکت تعریف شده که هر آرایه شامل یک ورودی (`List<Object>`) و مقدار مورد انتظار است.
- تست‌ها شامل نمونه‌های معنادار برای هر دو نوع داده `String` و `Integer` هستند.
- متد `testMinNormalCases()` متد `Min.min()` را فراخوانی کرده و با مقدار مورد انتظار مقایسه می‌کند.

این نوع تست باعث پوشش «Happy Path»ها با داده‌های متنوع شده و توسعه‌پذیری را در تست افزایش می‌دهد.

```java
import static org.junit.Assert.*;
import java.util.*;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

@RunWith(Parameterized.class)
public class MinParamTest {

    private List<Object> inputList;
    private Object expectedMin;

    public MinParamTest(List<Object> inputList, Object expectedMin) {
        this.inputList = inputList;
        this.expectedMin = expectedMin;
    }

    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {
            { Arrays.asList("cat"), "cat" },
            { Arrays.asList("dog", "cat"), "cat" },
            { Arrays.asList("zebra", "apple", "monkey"), "apple" },

            { Arrays.asList(42), 42 },
            { Arrays.asList(5, 10, 1, 9), 1 },
            { Arrays.asList(-5, 0, -1), -5 }
        });
    }

    @Test
    public void testMinNormalCases() {
        Object result = Min.min(inputList);
        assertEquals("Testing min of list: " + inputList, expectedMin, result);
    }
}
```

ح)

یکی از ابزارهای تست خودکار که در اسلایدها به آن اشاره‌ای نشده، اما بسیار محبوب و پرکاربرد در صنعت نرم‌افزار است، ابزار **Selenium** است.

معرفی Selenium

Selenium یک ابزار متن‌باز برای **تست خودکار نرم‌افزارهای مبتنی بر وب** است. این ابزار به توسعه‌دهندگان و تیم‌های QA اجازه می‌دهد تا تعاملات کاربر با مرورگر را شبیه‌سازی و بررسی کنند. برخلاف JUnit که عمدتاً برای تست واحد در برنامه‌های جاوا استفاده می‌شود، Selenium برای **تست سیستم**، **تست رابط کاربری (UI)** و **تست مرورگر محور (browser-based)** طراحی شده است.

اجزای اصلی Selenium

1. **Selenium WebDriver**
   این مؤلفه هسته‌ای‌ترین بخش Selenium است که با مرورگرها مستقیماً در تعامل است. WebDriver از طریق APIهای زبان‌های برنامه‌نویسی مختلف مانند C، Python، Java# و JavaScript قابل استفاده است.

2. **Selenium IDE (Integrated Development Environment)**
   افزونه‌ای برای مرورگرهای Chrome و Firefox که امکان ضبط و اجرای تست‌ها را بدون نیاز به کدنویسی فراهم می‌کند. برای افراد غیرتوسعه‌دهنده بسیار مناسب است.

3. **Selenium Grid**
   ابزاری برای اجرای تست‌های Selenium به‌صورت توزیع‌شده روی چند ماشین و مرورگر مختلف، مناسب برای اجرای تست‌های موازی (parallel).

4. **Selenium RC (Remote Control)** [غیرفعال شده]
   نسخه‌ی قدیمی‌تر Selenium که امروزه با WebDriver جایگزین شده است.

نحوه عملکرد Selenium WebDriver

در زیر یک نمونه‌ی ساده از تست یک فرم لاگین با Selenium در زبان Java ارائه شده است:

```java
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;

public class LoginTest {
    public static void main(String[] args) {
        System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");

        WebDriver driver = new ChromeDriver();

        driver.get("https://example.com/login");

        driver.findElement(By.name("username")).sendKeys("testuser");
        driver.findElement(By.name("password")).sendKeys("password123");

        driver.findElement(By.id("loginButton")).click();

        String expectedTitle = "Dashboard";
        String actualTitle = driver.getTitle();
        if (actualTitle.equals(expectedTitle)) {
            System.out.println("Login test passed.");
        } else {
            System.out.println("Login test failed.");
        }

        driver.quit();
    }
}
```

مزایای Selenium

- **متن‌باز و رایگان**
- پشتیبانی از مرورگرهای مختلف (Chrome، Firefox، Safari و...)
- قابل استفاده در زبان‌های مختلف برنامه‌نویسی
- قابلیت ادغام با ابزارهای CI/CD مانند Jenkins
- امکان اجرای تست‌های هم‌زمان با Selenium Grid

محدودیت‌ها

- فقط برای تست نرم‌افزارهای **وب‌محور** مناسب است.
- برای تست‌های پیچیده‌تر نیاز به مهارت برنامه‌نویسی دارد.
- برای بررسی UI ممکن است در مقایسه با ابزارهای سطح بالاتر مانند Cypress نیاز به تلاش بیشتری باشد.

منابع مورد استفاده:

1. Selenium Official Website
2. /ToolsQA Selenium Tutorial: https://www.toolsqa.com/selenium-webdriver
3. Guru99 Selenium Tutorial: https://www.guru99.com/selenium-tutorial.html
4. کتاب "Selenium Testing Tools Cookbook", Packt Publishing

ط)

برای پاسخ به این پرسش اختیاری و نشان دادن اینکه چرا بازنویسی متد `hashCode()` همراه با `equals()` ضروری است، ابتدا کلاس `Point` را با بازنویسی صرفاً متد `equals()` تعریف می‌کنیم، سپس با استفاده از `HashSet` مشکلی که در نبود `hashCode()` به‌وجود می‌آید را نشان خواهیم داد.

تعریف ناقص کلاس `Point` (بدون بازنویسی `hashCode`)

```java
import java.util.*;

public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Point)) return false;
        Point p = (Point) o;
        return this.x == p.x && this.y == p.y;
    }

}
```

استفاده از HashSet برای نمایش مشکل

```java
public class Main {
    public static void main(String[] args) {
        Point p1 = new Point(1, 2);
        Point p2 = new Point(1, 2);

        Set<Point> set = new HashSet<>();
        set.add(p1);
        set.add(p2);

        System.out.println("Size of set: " + set.size());
    }
}
```

توضیح: با وجود اینکه `p1.equals(p2)` مقدار `true` را برمی‌گرداند، `HashSet` برای تعیین یکتا بودن، ابتدا از `hashCode()` استفاده می‌کند. از آنجا که ما `hashCode()` را بازنویسی نکرده‌ایم، `p1` و `p2` هش‌کدهای متفاوتی دارند و در نتیجه هر دو به‌عنوان عناصر متفاوت در `HashSet` ذخیره می‌شوند.

اصلاح کلاس با بازنویسی `hashCode()`

```java
@Override
public int hashCode() {
    return 31 * x + y;
}
```

اجرای مجدد پس از اصلاح

```java
public class Main {
    public static void main(String[] args) {
        Point p1 = new Point(1, 2);
        Point p2 = new Point(1, 2);

        Set<Point> set = new HashSet<>();
        set.add(p1);
        set.add(p2);

        System.out.println("Size of set: " + set.size());
    }
}
```

نتیجه‌گیری

در Java، زمانی‌که `equals()` بازنویسی می‌شود، طبق قرارداد `Object` باید `hashCode()` نیز بازنویسی شود. در غیر این صورت، رفتار مجموعه‌هایی مانند `HashSet` نادرست خواهد بود، زیرا این ساختارها برای تشخیص تکرار، ابتدا از `hashCode()` و سپس از `equals()` استفاده می‌کنند.