ارشیا عطایی نایینی ۸۱۰۱۰۰۲۵۲
امیرپارسا موبد ۸۱۰۱۰۰۲۷۱

آزمون نرم افزار
تمرین چهارم

Github Reposiroty: https://github.com/ParsaMOBB/SWT-Course-Projects

Last Commit: 427e5fc1c7300107474ec792a0f4c0a0c068fda6

سوال اول:

تفاوت اصلی بین `@WebMvcTest` و `@SpringBootTest` در محدوده بارگذاری کانتکست برنامه است. وقتی از `@WebMvcTest` استفاده می‌کنیم، فقط بخش‌های مرتبط با لایه وب مانند کنترلرها، `ControllerAdvice` و ابزارهایی مثل `MockMvc` بارگذاری می‌شوند و سایر بخش‌های برنامه مثل لایه سرویس یا دیتابیس در تست حضور ندارند. این انوتیشن برای تست رفتار کنترلرها و اطمینان از عملکرد صحیح آنها طراحی شده است. در مقابل، `@SpringBootTest` کل کانتکست برنامه را بارگذاری می‌کند و به ما امکان می‌دهد تستی جامع از تمام بخش‌های برنامه داشته باشیم. این انوتیشن برای تست یکپارچه مناسب است و تضمین می‌کند که همه اجزای برنامه در کنار هم به درستی کار می‌کنند. با این حال، استفاده از آن به دلیل بارگذاری کل برنامه می‌تواند زمان بیشتری ببرد.

سوال دوم:

الف) $p = (\neg a \wedge b) \vee (b \wedge c) \vee (\neg b \wedge \neg c)$

	a	b	c	P
1	T	T	T	T
2	T	T	F	F
3	T	F	T	F
4	T	F	F	T
5	F	T	T	T

6	F	T	F	T
7	F	F	T	F
8	F	F	F	T

ب) ابتدا به ازای هر کلاز عباراتی را بدست می آوریم که با عوض کردن آن P عوض شود.

Clause a:

$((False \wedge b) \vee (b \wedge c) \vee (\neg b \wedge \neg c)) \oplus ((True \wedge b) \vee (b \wedge c) \vee (\neg b \wedge \neg c)) =$

$((b \wedge c) \vee (\neg b \wedge \neg c)) \oplus (b \vee (\neg b \wedge \neg c)) = b \wedge \neg c$

Clause b:

$((\neg a \wedge True) \vee (True \wedge c) \vee (False \wedge \neg c)) \oplus ((\neg a \wedge False) \vee (False \wedge c) \vee (True \wedge \neg c))$

$= (\neg a \vee c) \oplus (\neg c) = a \mid c$

Clause c:

$((\neg a \wedge b) \vee (b \wedge True) \vee (\neg b \wedge False)) \oplus ((\neg a \wedge b) \vee (b \wedge False) \vee (\neg b \wedge True))$

$= ((\neg a \wedge b) \vee b) \oplus ((\neg a \wedge b) \vee \neg b) = a \mid \neg b$

حال تعداد حالات True شدن نتیجه a برابر 1 حالت است و برای b برابر 3 حالت است و برای c برابر 3حالت است پس به ازای a داریم جفت سطر [2, 6] و به ازای b جفت سطرهای [[1, 3], [2, 4], [5, 7], [5, 3], [1, 7]] و به ازای c جفت سطرهای [[1, 2], [4, 3], [4, 7], [8, 3], [8, 7]].

پ) سطرهایی که همه کلازها جز کلاس اکتیو یکسان است جواب این مسئله هستند که برای a داریم [6, 2] و برای b داریم [[1, 3], [5, 7], [2, 4]] و برای c داریم [[1, 2], [3, 4], [7, 8]].
خیر اینگونه نیست زیرا به طور کلی RACC است که CACC را subsume میکند و همانطور که میبینیم همه موارد بخش ب در بخش پ وجود ندارند بلکه بلعکس، موارد بخش پ در ب موجود هستند.

ت) خیر، اگر a = F, b = T, c = T را در نظر بگیرید و a = T, b = F, c = F، در عبارت اول کلاز اول T و دومی T و سومی F میشود و عبارت کلی T و در دومی کلاز اول F دومی F و سومی T میشود و عبارت کلی همچنان T که شامل clause coverage است اما دارای predicate coverage نیست.

سوال سوم:

ابتدا بر حسب سه متغیر داد این‌ها را بلاک بندی میکنیم:

price:

- price <= 0
- 0 < price < minPurchase
- minPurchase <= price

minPurchase:

- minPurchase <= 0
- minPurchase > 0

discountRate:

- discountRate < 0
- discountRate == 0
- 0 < discountRate <= 1
- 1 < discountRate

حال اگر سه حال اول را A0, A1, A2 و دو حالت دوم را B0, B1 و سه حالت سوم را C0, C1, C2 در نظر بگیریم، در کل 18 حالت موجود است:

```
@Test
    public void testA0B0C0() {
        String result = calculateDiscountedPrice(-1, -1, -0.1);
        assertEquals("Invalid input", result);
    }

    @Test
    public void testA0B0C1() {
        String result = calculateDiscountedPrice(-1, -1, 0);
        assertEquals("Invalid input", result);
    }

    @Test
    public void testA0B0C2() {
        String result = calculateDiscountedPrice(-1, -1, 0.5);
        assertEquals("Invalid input", result);
    }

    @Test
    public void testA0B1C0() {
        String result = calculateDiscountedPrice(-1, 100, -0.1);
        assertEquals("Invalid input", result);
    }

    @Test
    public void testA0B1C1() {
        String result = calculateDiscountedPrice(-1, 100, 0);
        assertEquals("Invalid input", result);
    }

    @Test
    public void testA0B1C2() {
        String result = calculateDiscountedPrice(-1, 100, 0.5);
        assertEquals("Invalid input", result);
    }
```

```
@Test
public void testA1B0C0() {
    String result = calculateDiscountedPrice(50, -1, -0.1);
    assertEquals("Invalid input", result);
}

@Test
public void testA1B0C1() {
    String result = calculateDiscountedPrice(50, -1, 0);
    assertEquals("Invalid input", result);
}

@Test
public void testA1B0C2() {
    String result = calculateDiscountedPrice(50, -1, 0.5);
    assertEquals("Invalid input", result);
}

@Test
public void testA1B1C0() {
    String result = calculateDiscountedPrice(50, 100, -0.1);
    assertEquals("Invalid input", result);
}

@Test
public void testA1B1C1() {
    String result = calculateDiscountedPrice(50, 100, 0);
    assertEquals("50.0", result);
}

@Test
public void testA1B1C2() {
    String result = calculateDiscountedPrice(50, 100, 0.5);
    assertEquals("50.0", result);
}
```

```
@Test
public void testA2B0C0() {
    String result = calculateDiscountedPrice(150, -1, -0.1);
    assertEquals("Invalid input", result);
}

@Test
public void testA2B0C1() {
    String result = calculateDiscountedPrice(150, -1, 0);
    assertEquals("Invalid input", result);
}

@Test
public void testA2B0C2() {
    String result = calculateDiscountedPrice(150, -1, 0.5);
    assertEquals("Invalid input", result);
}

@Test
public void testA2B1C0() {
    String result = calculateDiscountedPrice(150, 100, -0.1);
    assertEquals("Invalid input", result);
}

@Test
public void testA2B1C1() {
    String result = calculateDiscountedPrice(150, 100, 0);
    assertEquals("150.0", result);
}

@Test
public void testA2B1C2() {
    String result = calculateDiscountedPrice(150, 100, 0.5);
    assertEquals("75.0", result);
}
```