مدرس: دکتر احسان خامس‌پناه
طراحان: علی هدایی، آوا میرمحمدمهدی، امیرعلی وحیدی

بخش تئوری

سوال اول

به صورت مختصر تفاوت بین دو انوتیشن @WebMvcTest و @SpringBootTest را بیان کنید.

پاسخ:

تفاوت اصلی بین دو انوتیشن @WebMvcTest و @SpringBootTest در هدف و محدوده آزمون آن‌ها است.

در @WebMvcTest، هدف آزمودن لایه کنترلرها است و تنها اجزای مربوط به وب، مانند کنترلرها را بارگذاری می‌کند. این آزمون معمولاً زمانی استفاده می‌شود که بخواهید کنترلرهای MVC خود را آزمون کنید بدون اینکه نیازی به بارگذاری کل محتوای برنامه یا تعامل با دیتابیس داشته باشید.

@SpringBootTest، برای آزمون‌های یکپارچه (Integration Testing) استفاده می‌شود؛ بدین صورت که یک آزمون جامع از راه‌اندازی، پیکربندی و اجزای برنامه شما ارائه می‌دهد و به صورت پیش‌فرض، کل محتوای Spring را بارگذاری می‌کند. این آزمون معمولاً برای آزمون کل جریان برنامه استفاده می‌شود تا اطمینان حاصل شود که همه اجزا (شامل کنترلرها، سرویس‌ها و ریپازیتوری‌ها)، به درستی با هم کار می‌کنند.

سوال دوم

با در نظر گرفتن predicate زیر به سوالات پاسخ دهید.

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

الف) جدول درستی را بنویسید.

ب) تمام جفت ردیف‌هایی که Correlated Active Clause Coverage (CACC) را برآورده می‌کند بنویسید.

پ) تمام جفت ردیف‌هایی که Restricted Active Clause Coverage (RACC) را برآورده می‌کند بنویسید.

آیا جفت‌های نوشته‌شده در قسمت ب، زیرمجموعه جفت‌های نوشته‌شده در این قسمت هستند؟ چرا؟

ت) در predicate داده‌شده، آیا برآورده کردن clause coverage می‌تواند predicate coverage را نتیجه دهد؟ درستی آن را نشان دهید یا برای آن مثال نقض بزنید.

پاسخ:

الف)

Row	a	b	c	$\neg a \wedge b$	$b \wedge c$	$\neg b \wedge \neg c$	p
1	T	T	T	F	T	F	T
2	T	T	F	F	F	F	F
3	T	F	T	F	F	F	F
4	T	F	F	F	F	T	T
5	F	T	T	T	T	F	T
6	F	T	F	T	F	F	T
7	F	F	T	F	F	F	F
8	F	F	F	F	F	T	T

ب)

major clause : a

$$P_a = P_{a=True} \oplus P_{a=False}$$
$$= ((F \wedge b) \vee (b \wedge c) \vee (\neg b \wedge \neg c)) \oplus ((T \wedge b) \vee (b \wedge c) \vee (\neg b \wedge \neg c))$$
$$= ((b \wedge c) \vee (\neg b \wedge \neg c)) \oplus (b \vee (b \wedge c) \vee (\neg b \wedge \neg c))$$
$$= (b \leftrightarrow c) \oplus (b \vee \neg c)$$

اگر جدول درستی عبارت بدست آمده را بکشیم، ساده شده‌ی P_a برابر با مقدار زیر خواهد بود:

$$P_a = b \wedge \neg c$$
$$(2, 6)$$

---

major clause : b

$$P_b = P_{b=True} \oplus P_{b=False}$$
$$= ((\neg a \wedge T) \vee (T \wedge c) \vee (F \wedge \neg c)) \oplus ((\neg a \wedge F) \vee (F \wedge c) \vee (T \wedge \neg c))$$
$$= (\neg a \vee c) \oplus (\neg c)$$
$$= a \vee (\neg a \wedge c)$$

حال باید از بین حالاتی که $a = true$ یا $(\neg a \wedge c) = true$ است، جفت‌هایی را انتخاب کنیم که p با مقادیر b متفاوت، پاسخ متفاوتی داشته باشد که شامل جفت‌هایی زیر است:

(2, 4) و هر حالت حاصل ضرب (3, 7) و (1, 5)

{(1, 3), (1, 7), (3, 5), (5, 7)}, (2, 4)

---

major clause : c

$$P_c = P_{c=True} \oplus P_{c=False}$$
$$= ((\neg a \wedge b) \vee (b \wedge T) \vee (\neg b \wedge F)) \oplus ((\neg a \wedge b) \vee (b \wedge F) \vee (\neg b \wedge T))$$
$$= ((\neg a \wedge b) \vee b) \oplus ((\neg a \wedge b) \vee \neg b)$$
$$= ((\neg a \wedge b) \vee b) \oplus ((\neg a \wedge b) \vee \neg b)$$
$$= (b) \oplus (\neg a \vee \neg b)$$
$$= a \vee (\neg a \wedge \neg b)$$

حال باید از بین حالاتی که $a = true$ یا $(\neg a \wedge \neg b) = true$ است، جفت‌هایی را انتخاب کنیم که p با مقادیر c متفاوت، پاسخ متفاوتی داشته باشد که شامل جفت‌های زیر است:

(1, 2) و هر حالت حاصل ضرب (3, 7) و (4, 8)

(1, 2), {(3, 4), (3, 8), (4, 7), (7, 8)}

پ)

در اینجا باید جفت‌هایی را در نظر بگیریم که مقادیر minor clause ها به ازای تغییر در major clause ها ثابت باقی بماند.

major clause : a

(2, 6)

---

major clause : b

(1, 3), (2, 4), (5, 7)

---

major clause : c

(1, 2), (3, 4), (7, 8)

خیر هر جفتی که در CACC وجود دارد لزوما در RACC وجود ندارد ولی تمام جفت‌هایی که برای برآورده کردن RACC لازم هستند، شروط لازم برای CACC را برآورده می‌کنند.

ت)

خیر، برای مثال انتخاب ردیف 1 و 8، clause coverage را برآورده می‌کند ولی p در هر دوی آنها مقدار true دارد.

سوال سوم

با توجه به تابع زیر که قیمت نهایی یک خرید را پس از اعمال تخفیف بر اساس قیمت کالا، نرخ تخفیف و حداقل خرید محاسبه می‌کند، ورودی‌های آن را به بلاک‌هایی تقسیم کرده و سپس پوشش pair-wise را

پیاده‌سازی کنید. برای هر مورد آزمون (testcase)، با توجه به مسئله و مقادیر داده شده، assertion مناسب را انجام دهید.

```
public static String calculateDiscountedPrice(double price, double discountRate,
double minPurchase) {
    if (price <= 0 || discountRate < 0 || discountRate > 1 || minPurchase <= 0)
{
        return "Invalid input";
    } else if (price < minPurchase) {
        return String.valueOf(price);
    } else {
        double discountedPrice = price * (1 - discountRate);
        return String.valueOf(discountedPrice);
    }
}
```

پاسخ:

تحلیل ورودی‌ها و بلاک‌بندی:

- *price*: می‌تواند مقداری مثبت باشد یا صفر و کمتر از صفر (در این حالت باید خطا تولید شود).
- *discountRate*: باید بین 0 و 1 باشد. اگر کمتر از 0 یا بیشتر از 1 باشد، ورودی نامعتبر است.
- *minPurchase*: می‌تواند مقداری مثبت باشد، و اگر برابر یا کمتر از صفر باشد، باید خطا برگردانده شود.

پوشش Pair-wise:

برای پوشش تمام حالات ترکیبی (pair-wise)، آزمون‌هایی را به این شکل تنظیم می‌کنیم:

- (price > minPurchase, discountRate = 0, minPurchase > 0)
- (price > minPurchase, discountRate = 0.5, minPurchase > 0)
- (price > minPurchase, discountRate = 1, minPurchase > 0)
- (price <= 0, discountRate = 0, minPurchase > 0)
- (price > 0, discountRate < 0, minPurchase > 0)
- (price > minPurchase, discountRate > 1, minPurchase > 0)
- (price < minPurchase, discountRate > 0, minPurchase > 0)
- (price > minPurchase, discountRate = 0, minPurchase <= 0)

پیاده‌سازی کد آزمون و assertion:

```
@Test
public void testCalculateDiscountedPrice() {
    // Case 1: price > minPurchase, discountRate = 0, minPurchase > 0
```

```
    assertEquals("100", calculateDiscountedPrice(100, 0, 50), "Test Case 1
Failed");

    // Case 2: price > minPurchase, discountRate = 0.5, minPurchase > 0
    assertEquals("50", calculateDiscountedPrice(100, 0.5, 50), "Test Case 2
Failed");

    // Case 3: price > minPurchase, discountRate = 1, minPurchase > 0
    assertEquals("0", calculateDiscountedPrice(100, 1, 50), "Test Case 3
Failed");

    // Case 4: price <= 0, discountRate = 0, minPurchase > 0
    assertEquals("Invalid input", calculateDiscountedPrice(0, 0, 50), "Test Case
4 Failed");

    // Case 5: price > 0, discountRate < 0, minPurchase > 0
    assertEquals("Invalid input", calculateDiscountedPrice(100, -0.1, 50), "Test
Case 5 Failed");

    // Case 6: price > minPurchase, discountRate > 1, minPurchase > 0
    assertEquals("Invalid input", calculateDiscountedPrice(100, 1.1, 50), "Test
Case 6 Failed");

    // Case 7: price < minPurchase, discountRate > 0, minPurchase > 0
    assertEquals("30", calculateDiscountedPrice(30, 0.2, 50), "Test Case 7
Failed");

    // Case 8: price > minPurchase, discountRate = 0, minPurchase <= 0
    assertEquals("Invalid input", calculateDiscountedPrice(100, 0, -10), "Test
Case 8 Failed");
}
```

این کد آزمون، هشت حالت مختلف ورودی‌ها را پوشش می‌دهد و اطمینان حاصل می‌کند که تابع *calculateDiscountedPrice* در هر حالت، پاسخ مناسب را ارائه می‌دهد.