

ریپو گیت هاب

Repository: <https://github.com/elahekhodaverdi/SWT-Fall103>

Commit Hash: b417706115c2ae9d553f7de777009af3787b23e6

باگ های حل شده در کد

در کلاس TableController در تابع addTable هنگام انجام عملیات parseInt به دنبال پارامتر seatNumber در ورودی می گردیم. این در حالی است که این فیلد seatsNumber نام دارد. برای حل این مشکل کافی است که کد این متد را به نحو زیر تغییر دهیم.

```
int seatsNumber;
try {
    seatsNumber = Integer.parseInt(params.get("seatsNumber"));
} catch (Exception ex) {
    throw new ResponseException(HttpStatus.BAD_REQUEST, PARAMS_BAD_TYPE);
}
```

و با این تست متوجه این باگ شدیم:

```
@Test
void testAddTableWithSuccess() throws Exception {
    Map<String, String> params = new HashMap<>();
    params.put("seatsNumber", "4");

    when(restaurantService.getRestaurant(Mockito.anyInt())).thenReturn(restaurant);

    mockMvc.perform(post("/tables/{restaurantId}", restaurant.getId())
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(params)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.message").value("table added"));
}
```

سوال اول

تفاوت اصلی بین انوتیشن های **SpringBootTest** و **WebMvcTest** در میزان integration testing ای است که این دو ارائه می دهند. انوتیشن **SpringBootTest** تمام کانتکست برنامه را بارگذاری می کند، که شامل پایگاه داده، امنیت، و سایر اجزای زیرساختی است. به همین دلیل، برای این نوع از تست مناسب تر است، چرا که می خواهیم بررسی کنیم چگونه لایه های مختلف برنامه با یکدیگر تعامل دارند.

در مقابل، **WebMvcTest** بیشتر برای تست لایه کنترلرها مناسب است، زیرا تنها لایه وب را بارگذاری می‌کند و سایر لایه‌ها را شبیه‌سازی می‌کند. این انوتیشن برای تست واحد رفتار کنترلرها استفاده می‌شود، بدون اینکه نیاز به نگرانی در مورد پیچیدگی‌های سایر لایه‌ها باشد. این روش به طور کلی به unit testing کنترلرها می‌پردازد.

یکی دیگر از تفاوت‌های مهم این دو انوتیشن، میزان configuration مورد نیاز است. از آنجایی که **SpringBootTest** تمام کانتکست برنامه را بارگذاری می‌کند، ممکن است نیاز به config اضافی برای تست‌ها داشته باشیم، مانند config پایگاه داده یا تنظیمات امنیتی.

اما **WebMvcTest** نیاز به configuration کمتری دارد، زیرا فقط لایه وب بارگذاری می‌شود. به این ترتیب، برای نوشتن و نگهداری تست‌ها ساده‌تر است و نیازی به config اجزای دیگر لایه‌های برنامه ندارید.

با این توصیف **WebMvcTest** سریع‌تر اجرا شده و برخلاف **SpringBootTest** نیاز به راه اندازی manual کمتری دارد.

سوال دوم

$$P = (\sim a \ \& \ b) \mid (b \ \& \ c) \mid (\sim b \ \& \ \sim c)$$

(الف)

| Clause a | Clause b | Clause c | predicate |
|----------|----------|----------|-----------|
| T | T | T | T |
| T | F | T | F |
| T | T | F | F |
| T | F | F | T |
| F | T | T | T |
| F | F | T | F |
| F | T | F | T |
| F | F | F | T |

(ب)

| id | Clause a | Clause b | Clause c | predicate |
|----|----------|----------|----------|-----------|
| 1 | T | T | T | T |
| 2 | T | F | T | F |
| 3 | T | T | F | F |
| 4 | T | F | F | T |
| 5 | F | T | T | T |
| 6 | F | F | T | F |
| 7 | F | T | F | T |
| 8 | F | F | F | T |

اگر a به عنوان major clause باشد: (7,3)

اگر b به عنوان major clause باشد:

زوج های زیر

(1,2) (1,6) (3, 4) (5,2) (5,6)

اگر c به عنوان major clause باشد:

زوج های زیر

(2,4) (2,8) (3,1) (6,4) (6,8)

(ج)

| Clause a | Clause b | Clause c | predicate | C_a | C_b | C_c |
|----------|----------|----------|-----------|-----|-----|-----|
| T | T | T | T | | 2, | 5 |
| T | F | T | F | | 2, | 6 |
| T | T | F | F | 1, | 3, | 5 |
| T | F | F | T | | 3, | 6 |

| | | | | | | |
|---|---|---|---|----|----|---|
| F | T | T | T | | 4, | |
| F | F | T | F | | 4, | 7 |
| F | T | F | T | ,1 | | |
| F | F | F | T | | | 7 |

جفت های مربوط به این قسمت در جدول با آیدی مشخص شدند و هر جفت یک شماره یکسان دریافت کرده است. لذا از نوشتن دوباره آن ها به صورت جداگانه و دوباره اجتناب کردیم.

خیر جفت های نوشته شده در قسمت ب زیرمجموعه جفت های نوشته شده در این قسمت نیستند زیرا در قسمت پ ما تنها حالاتی را در نظر میگیریم که بقیه clause ها دقیقا یکسان باشند ولی در حالت الف می توانند متفاوت باشند پس شامل جفت های بیشتری است و در واقع برعکس است. جفت های این قسمت زیرمجموعه ای از جفت های قسمت ب هستند.

البته باید توجه داشت که RACC به طوری است که CACC را subsume می کند.

(ت)

تنها از یک حالت می توان با clause coverage به predicate coverage دست یافت آن هم با در نظر گرفتن دو تست زیر است:

FTF

TFT

در هیچ یک از بقیه ترکیب ها نمی توان به predicate coverage دست یافت. دلیل آن هم به این صورت است که اگر در ترکیب در نظر گرفته شده b و c یکسان باشند آنگاه در هر دو تست مربوط به clause coverage ما predicate را true می کنیم. اگر b و c متفاوت باشند دو حالت می مانند که آنگاه براساس predicate تنها در یک حالت predicate coverage خواهیم داشت. پس clause coverage لزوما باعث predicate coverage نمی شود. مثال نقض:

TTT -> T

FFF -> T

سوال سوم

کد زیر را داریم:

```
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);
}
}

```

برای کد بالا سه پارامتر داریم که در ادامه block های مربوط به هریک را عنوان می‌کنیم.

:Price

1. A1: مقدار آن کوچک‌تر از صفر باشد (invalid).
2. A2: مقدار آن مساوی صفر باشد (invalid).
3. A3: مقدار آن بزرگتر از صفر باشد (valid).

:discountRate

1. B1: مقدار آن کمتر صفر باشد (invalid).
2. B2: مقدار آن مساوی صفر باشد (valid).
3. B3: مقدار آن بین صفر و یک باشد (valid).
4. B4: مقدار آن مساوی یک باشد (valid).
5. B5: مقدار آن بزرگتر از یک باشد (invalid).

:minPurchase

1. C1: مقدار آن کوچک‌تر از صفر باشد (invalid).
2. C2: مقدار آن مساوی صفر باشد (invalid).
3. C3: مقدار آن بزرگتر از صفر باشد (valid).

برای حالت else if نیز باید یک characteristics نیز داشته باشیم که $price < minPurchase$ را در نظر بگیرد. حالت $price < minPurchase$ را D1 و حالت برعکس آن را D2 در نظر می‌گیریم.

برای PWC تست های زیر را خواهیم داشت.

تست‌ها:

| | | |
|---------------|----------------|----------------|
| A1,B1, C1, D2 | A2, B1, C2, D2 | A3, B1, C3, D1 |
| A1,B2, C2, D1 | A2, B2, C3, D1 | A3, B2, C1, D2 |
| A1,B3, C3, D1 | A2, B3, C1, D2 | A3, B3, C2, D2 |
| A1,B4, C1, D1 | A2, B4, C2, D2 | A3, B4, C3, D2 |
| A1,B5, C1, D2 | A2, B5, C2, D2 | A3, B5, C3, D1 |

حالات مربوط به A3, C1, D1 و A3, C2, D1 و A2, C1, D1 و A2, C2, D1 و A1, C2, D2 و A1, C3, D2 و A2, C3, D2 غیر ممکن هستند و نمی‌توانیم آن‌ها را داشته باشیم برای همین در جدول نیز عنوان نشده‌اند.
تست‌ها:

```
import org.junit.jupiter.api.Test;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;

class DiscountCalculatorTest {

    private final List<Double> A = List.of(-10.0, 0.0, 50.0);
    private final List<Double> B = List.of(-0.1, 0.0, 0.2, 1.0, 1.1);
    private final List<Double> C = List.of(-5.0, 0.0, 30.0);

    @Test
    void testInvalidInputs() {
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(0), B.get(0), C.get(0))); //
A1, B1, C1, D2
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(1), B.get(0), C.get(1))); //
A2, B1, C2, D2
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(2), B.get(0), C.get(2))); //
A3, B1, C3, D1
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(0), B.get(1), C.get(1))); //
A1, B2, C2, D1
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(1), B.get(1), C.get(2))); //
A2, B2, C3, D1
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(2), B.get(1), C.get(0))); //
A3, B2, C1, D2
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(0), B.get(2), C.get(2))); //
A1, B3, C3, D1
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(1), B.get(2), C.get(0))); //
A2, B3, C1, D2
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(2), B.get(2), C.get(1))); //
A3, B3, C2, D2
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(0), B.get(3), C.get(0))); //
A1, B4, C1, D1
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(1), B.get(3), C.get(1))); //
A2, B4, C2, D2
```

```
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(0), B.get(4), C.get(0))); //
A1, B5, C1, D2
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(1), B.get(4), C.get(1))); //
A2, B5, C2, D2
        assertEquals("Invalid input",
DiscountCalculator.calculateDiscountedPrice(A.get(2), B.get(4), C.get(2))); //
A3, B5, C3, D1
    }

    @Test
    void testValidInputs() {
        assertEquals("0.0", DiscountCalculator.calculateDiscountedPrice(A.get(2),
B.get(3), C.get(2))); // A3, B4, C3, D2
    }
}
```