گزارش‌کار تمرین کامپیوتری سوم آزمون نرم افزار

محمدهادی بابالو – 810199380 سنا ساری نوایی – 810199435

شناسه آخرین کامیت

https://github.com/Hadi-loo/Software-Testing-Course/commit/f4fce844b4d4ca0f903aa729f74495d5af2fba29

سوال 1

- Statement Coverage: این تکنیک تضمین می‌کند که تمام statement های کد حداقل یک بار اجرا می‌شوند.
- Branch Coverage: این تکنیک تضمین می‌کند که تمام branch های کد حداقل یک بار اجرا می‌شوند یا خیر.

در تابع داده شده، امکان پوشش جمله 100 درصدی با استفاده از یک آزمایه امکان‌پذیر نیست؛ زیرا در اینجا یک شرط قرار گرفته است که در صورت اجرای آن، شرط درون if اجرا شده و return صورت می‌گیرد. اگر شرط برقرار نباشد نیز return خارج از شرط if انجام می‌گیرد. یعنی یا بدنه if اجرا می‌شود و یا return انتهای تابع اجرا می‌شود. در هیچ حالتی امکان پوشش جمله 100 درصدی را با یک آزمایه نداریم. امکان پوشش شاخه 100 درصدی نیز در اینجا امکان‌پذیر نیست؛ زیرا اگر شرط درست باشد، شاخه true و در غیر اینصورت شاخه false اجرا می‌شود. Branch coverage به تست هر دو شاخه نیاز دارد که در یک آزمایه امکان‌پذیر نیست. پس در این حالت، هیچکدام از آن دو نمی‌توانند به 100 درصد برسند.

```
public boolean equals (Object obj) {
    if (obj instanceof Order order) {
        return id == order.id;
    }
    return false;
}
```

در حالت دوم می‌توانیم پوشش جمله 100 درصدی داشته باشیم اما پوشش شاخه 100 درصدی امکان‌پذیر نیست که این همان خواسته سوال است. در صورتی‌که آزمایه‌ای تعریف کنیم که شرط if در آن صدق کند، از تمامی statement ها گذر کرده و آنها را پوشش می‌دهد. اما پوشش شاخه همچنان 50 درصد باقی می‌ماند، زیرا باید در تست حالتی که شرط درست نباشد را هم امتحان کنیم تا پوشش شاخه کامل شود.

```
public boolean equals (Object obj) {
    var result = false;
    if (obj instanceof Order order) {
        result = id == order.id;
    }
    return result;
}
```

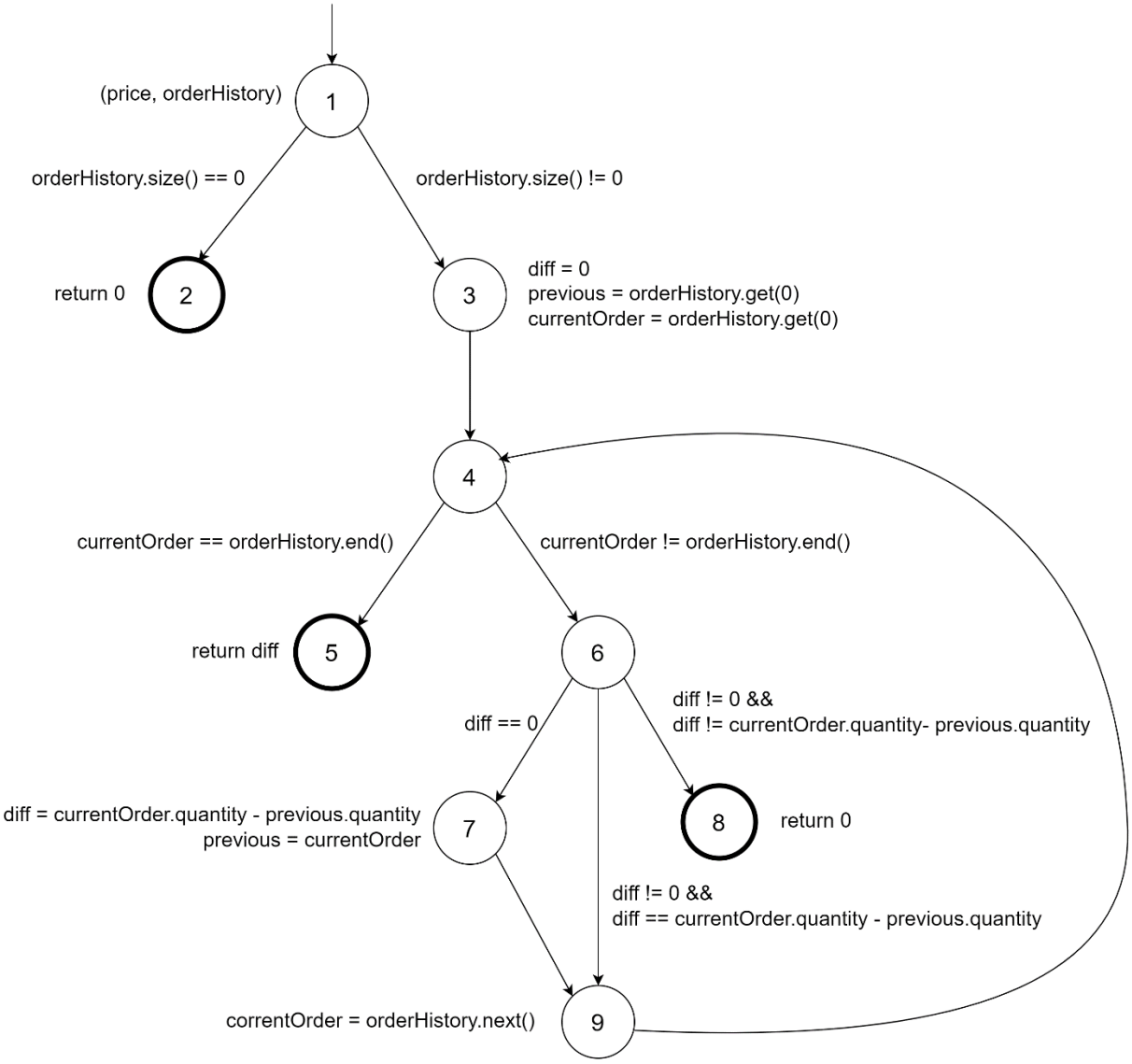
در تست زیر می‌توانیم به پوشش جمله 100 درصدی برسیم. در این تست به این گونه عمل کردیم که obj یک instance از کلاس Order می‌باشد؛ در نتیجه شرط if برقرار است و وارد آن می‌شویم. Result مقدار true می‌گیرد و در نهایت برگردانده می‌شود.

```
@Test
@DisplayName("equals should return true when the id is the same")
public void testEqualsReturnsTrueWhenIdIsSame() {
    Order order = new Order();
    order.setId(1);
    Order order2 = new Order();
    order2.setId(1);
    assertTrue(order.equals(order2));
```

محمدهادی بابالو – 810199380 سنا ساری نوایی – 810199435

سوال 2

Control flow graph در شکل زیر آمده است.



- **Prime path**

برای نوشتن prime path، ابتدا تمامی simple path ها را نوشتیم. پس از آن، مسیرهایی که subpath مسیر دیگری بودند را حذف کردیم تا در نهایت تمامی مسیر هایی به وجود بیاید که subpath نیستند. Simple path و prime path در زیر آورده شده است.

Simple path:

Length	Simple Path
0	[1], [2], [3], [4], [5], [6], [7], [8], [9]
1	[1, 2]!, [1, 3], [3, 4], [4, 5]!, [4, 6], [6, 7], [6,8]!, [6,9], [7,9], [9,4]
2	[1, 3, 4], [3, 4, 5]!, [3, 4, 6], [4, 6, 7], [4, 6,8]!, [4, 6, 9], [6, 7, 9], [6, 9, 4], [7, 9, 4], [9, 4, 5]!, [9, 4, 6]
3	[1, 3, 4, 5]!, [1, 3, 4, 6], [3, 4, 6, 7], [3, 4, 6, 8]! [3, 4, 6, 9], [4, 6, 7, 9], [4, 6, 9, 4]*, [6, 7, 9, 4] [6, 9, 4, 5]!, [6, 9, 4, 6]*, [7, 9, 4, 5]!, [7, 9, 4, 6], [9, 4, 6, 7], [9, 4, 6, 8]!, [9, 4, 6, 9]*
4	[1, 3, 4, 6, 7], [1, 3, 4, 6, 8]!, [1, 3, 4, 6, 9], [3, 4, 6, 7, 9], [4, 6, 7, 9, 4]*, [6, 7, 9, 4, 5]!, [6, 7, 9, 4, 6]*, [7, 9, 4, 6, 7]*, [7, 9, 4, 6, 8]! [9, 4, 6, 7, 9]*
5	[1, 3, 4, 6, 7, 9]

Prime path:

Length	Prime Path
0	-
1	[1, 2]!
2	-
3	[1, 3, 4, 5]!, [4, 6, 9, 4]*, [6, 9, 4, 5]!, [6, 9, 4, 6]*, [9, 4, 6, 9]*

4	[1, 3, 4, 6, 8]!, [1, 3, 4, 6, 9], [4, 6, 7, 9, 4]*, [6, 7, 9, 4, 5]!, [6, 7, 9, 4, 6]*, [7, 9, 4, 6, 7]*, [7, 9, 4, 6, 8]!, [9, 4, 6, 7, 9]*
5	[1, 3, 4, 6, 7, 9]

- **Du path**

برای نوشتن du path، ابتدا du pair ها را به دست آوردیم. نحوه بدست آوردن آن‌ها به این صورت است که باید last def و first use در هر متغیر را به دست آوریم. بعد از به دست آوردن جفت‌ها، یک مسیر از استیتی که متغیر مورد نظر در آن تعریف شده است، تعریف می‌کنیم. استیت پایان این مسیر، مکانی است که از آن متغیر استفاده می‌شود.

Identifier	Du Pair	Du Path
price	(1, -)	-
orderHistory	(1, (1, 2)) (1, (1, 3)) (1, 3) (1, (4, 5)) (1, (4, 6)) (1, 9)	[1, 2] [1, 3] [1, 3] [1, 3, 4, 5] [1, 3, 4, 6] [1, 3, 4, 6, 9] [1, 3, 4, 6, 7, 9]
diff	(3, 5) (3, (6, 7)) (3, (6, 8)) (3, (6, 9)) (7, (6, 7)) (7, (6, 8)) (7, (6, 9))	[3, 4, 5] [3, 4, 6, 7] [3, 4, 6, 8] [3, 4, 6, 9] [7, 9, 4, 6, 7] [7, 9, 4, 6, 8] [7, 9, 4, 6, 9]
previous	(3, 7) (3, (6, 8)) (3, (6, 9))	[3, 4, 6, 7] [3, 4, 6, 8] [3, 4, 6, 9]

	(7, 7) (7, (6, 8)) (7, (6, 9))	[7, 9, 4, 6, 7] [7, 9, 4, 6, 8] [7, 9, 4, 6, 9]
currentOrder	(3, (4, 5)) (3, (4, 6)) (3, 7) (3, (6, 8)) (3, (6, 9)) (9, (4, 5)) (9, (4, 6)) (9, 7) (9, (6, 8)) (9, (6, 9))	[3, 4, 5] [3, 4, 6] [3, 4, 6, 7] [3, 4, 6, 8] [3, 4, 6, 9] [9, 4, 5] [9, 4, 6] [9, 4, 6, 7] [9, 4, 6, 8] [9, 4, 6, 9]

سوال 3

بله ممکن است که test suite ای داشته باشیم که تمام du path ها را پوشش دهد ولی تمام prime path ها را نه، ولی برعکس آن ممکن نیست زیرا که prime path coverage، معیار all du paths را subsume می‌کند.

به طور مثال قطعه کد زیر را نگاه کنید:

```
int foo(int x) {
    int y = 0;
    if (x > 0) {
        y = x;
    } else {
        int z = 5;
    }
    return y;
}
```

در اینجا اگر test suite ای داشته باشیم که در آن x بزرگتر از صفر باشد و در نتیجه ما مسیری که وارد if شود را طی کنیم، تمام du path های ما پوشش داده می‌شود، اما همانطور که مشخص است تمام prime path ها پوشش

داده نشده‌اند و شاخه else و statement های در بدنه آن که عضوی از prime path هستند پوشش داده نمی‌شوند.

```
@Test
void fooTest() {
    assertEquals(10, foo(10))
}
```

سناریوی خواسته شده در صورت سوال با test suite بالا برآورده می‌شود.

سوال 4

با اینکه پوشش تمام prime path ها منجر به پوشش دادن تمام du path ها نیز می‌شود و محاسبه du path ها کار پرهزینه‌تری است، ولی باز هم مواردی وجود دارد که استفاده از معیار all du paths می‌تواند مفید باشد.

- در عمل وقتی که تمرکز ما روی جریان داده در برنامه است و اینکه چطور داده در برنامه هندل می‌شود می‌توان از all du path coverage استفاده کرد. این معیار باعث می‌شود که از اینکه تمام متغیرها قبل از استفاده شدنشان تعریف شده‌اند اطمینان حاصل کنیم و همچنین می‌تواند برای تشخیص مواردی مثل متغیرهای استفاده نشده هم به ما کمک کند.
- در برنامه‌های بزرگ و پیچیده که منجر به گراف‌های بسیار بزرگ می‌شوند، ممکن است که پوشش دادن تمام prime path ها به دلیل سایز زیاد آنها و محدودیت منابع ما عملا ناممکن باشد. در این موارد ممکن است که معیار all du paths بتواند تست‌های هدفمندتر و کوچک‌تری برای ما فراهم کند که بیشتر روی هدف‌های اصلی تست ما تمرکز دارند.
- درست است که محاسبه du path ها هزیه بیشتری برای ما دارد، اما در مواردی این معیار منجر به test suite های کوچک‌تر و بهینه‌تری می‌شود که مخصوصا در مواردی که جریان داده از control flow برایمان مهم‌تر است می‌تواند مفید واقع شود.