



آزمایشگاه طراحی نرم افزار

تمرین سوم

شایان طلایی ۹۸۱۰۹۵۴۸ محمدرضا ایجی ۹۸۱۰۱۱۹۳

بهار ۱۴۰۲

گزارش آزمایش

در این آزمایش به تست برنامه Jason-simple برای کد و دیکود کردن objectها به رشته Json پرداختیم. هدف ما در این آزمایش بالا بردن coverage تست برنامه در تعداد خط و تعداد توابع همه کلاسها بود. برای این منظور سه تابع تست جداگانه نوشتیم که edge caseهای functionalityهای برنامه را به صورت end-to-end تست کند. میزان coverage هر یک از توابع در ابتدا به صورت زیر بوده است.

Coverage: TestJson x			
Element	Class, %	Method, %	Line, %
org.json.simple	71% (5/7)	32% (30/91)	25% (204/786)
parser	75% (3/4)	40% (20/49)	33% (166/496)
ContainerFactory	100% (0/0)	100% (0/0)	100% (0/0)
ContentHandler	100% (0/0)	100% (0/0)	100% (0/0)
JSONParser	100% (1/1)	40% (6/15)	14% (37/251)
ParseException	0% (0/1)	0% (0/10)	0% (0/22)
Yylex	100% (1/1)	59% (13/22)	62% (124/200)
Yytoken	100% (1/1)	50% (1/2)	21% (5/23)
JSONArray	100% (1/1)	20% (5/25)	11% (19/161)
JSONAware	100% (0/0)	100% (0/0)	100% (0/0)
JSONObject	0% (0/1)	0% (0/9)	0% (0/36)
JSONStreamAware	100% (0/0)	100% (0/0)	100% (0/0)
JSONValue	100% (1/1)	62% (5/8)	20% (19/93)

برای ساده سازی تعاریف تستها کلاسی جدید به نام TestCase داخل توابع تستمان تعریف کردیم و به کمک آن تستها و جوابهای مورد انتظارمان را تعریف کردیم.

```
class TestCase {
    final String rawJson;
    final Object expected;

    public TestCase(String rawJson, Object expected) {
        this.rawJson = rawJson;
        this.expected = expected;
    }
}
```

Parsing tests for valid objects

برای تست کردن توابع parse روی داده‌های صحیح، تابع زیر را پیاده سازی کردیم.

```
@Test
public void testParseWhenGivenValidJSONStringThenReturnsObjectAndDoesNotThrowAnyException() {
    class TestCase {...}

    List<TestCase> testCases = Arrays.asList(
        // simple arrays
        new TestCase( rawJson: "[1, 2, 3]", JSONArray.of(1L, 2L, 3L)),
        new TestCase( rawJson: "[\"a\", \"bbb\"]", JSONArray.of("a", "bbb")),
        new TestCase( rawJson: "[1.01, 22.004]", JSONArray.of(1.01, 22.004)),

        // simple object
        new TestCase(
            rawJson: "{ \"a\": false, \"bb\" : 20, \"ddd\": null }",
            JSONObject.of("a", false, "bb", 20L, "ddd", null)),

        // nested object
        new TestCase(
            rawJson: "{ \"aa\": [1.0, 2.1, 3.14], \"bb\" : { \"ccc\": \"text\", \"ddd\": [\"t1\", \"t2\"] } }",
            JSONObject.of(
                "aa", JSONArray.of(1.0, 2.1, 3.14),
                "bb",
                JSONObject.of(
                    "ccc", "text",
                    "ddd", JSONArray.of("t1", "t2")
                )
            )
        ),

        // empty and null values
        new TestCase( rawJson: "[]", new JSONArray()),
        new TestCase( rawJson: "{}", new JSONObject()),
        new TestCase( rawJson: "null", expected: null),

        // single values
        new TestCase( rawJson: "100", expected: 100L),
        new TestCase( rawJson: "\"abc\"", expected: "abc"),
        new TestCase( rawJson: "10.10", expected: 10.10),
        new TestCase( rawJson: "false", expected: false),
        new TestCase( rawJson: "true", expected: true)
    );
};
```

همانطور که در بالای هر سری از تست‌ها کامنت شده است، در این کلاس فرایند parse شدن آرایه‌ها، آبجکت‌های ساده مثل رشته و عدد، دیکشنری‌های تو در تو، مقادیر خالی، و تک مقادیر را مورد آزمون قرار دادیم. برای بررسی کردن assert‌های درستی تست‌ها از توابع assertThatCode روی تست‌هایمان استفاده کردیم. با این کار تست‌های مختلف را به سادگی بررسی کردیم.

```
for (final TestCase testCase : testCases) {
    Object got = JSONValue.parse(testCase.rawJson);
    assertThat(got).isEqualTo(testCase.expected);
    assertThatCode(() -> JSONValue.parseWithException(testCase.rawJson)).doesNotThrowAnyException();
    assertThatCode(() -> JSONValue.parseWithException(new StringReader(testCase.rawJson))).doesNotThrowAnyException();
}
```

با افزودن این تست coverage برنامه به مقادیر زیر افزایش یافت.

JSONArray	100% (1/1)	11% (3/26)	2% (5/176)
JSONObject	100% (1/1)	30% (3/10)	19% (9/47)
JSONValue	100% (1/1)	50% (4/8)	7% (8/114)
JSONParser	100% (1/1)	66% (10/15)	36% (99/273)
ParseException	0% (0/1)	0% (0/10)	0% (0/28)
Yylex	100% (1/1)	63% (14/22)	68% (146/212)
Yytoken	100% (1/1)	50% (1/2)	25% (6/24)

Parsing tests for invalid objects

در این بخش ما حالتی را تست کردیم که انتظار می‌رود عمل parse کردن انجام نشود و مقدار null برگردد و تابع parseException برگرداند. همان طور که در تصویر می‌بینید حالتی مثل رشته خالی، براکت بسته نشده در لیست و دیکشنری، و کاراکترهای غیرمنتظره را تست کردیم.

```
@Test
public void testParseWhenGivenInvalidJSONStringThenReturnsNullAndThrowsException() {
    List<String> invalidJSONStrings = Arrays.asList(
        // empty
        "",

        // unclosed brackets
        "[1, [2]", "[",

        // unclosed braces
        "{ 1, {1:2}", "{",

        // unexpected characters
        "[1,,", "[{}]", "{1, 2, 3]",
        "1, test", "[1, hello]", "{1,2}"
    );

    for (final String invalidString : invalidJSONStrings) {
        assertThat(JSONValue.parse(invalidString)).isNull();
        assertThatThrownBy(() -> JSONValue.parseWithException(invalidString))
            .isInstanceOf(ParseException.class)
            .extracting(Throwable::getMessage)
            .asString()
            .isNotBlank();
        assertThatThrownBy(() -> JSONValue.parseWithException(new StringReader(invalidString)))
            .isInstanceOf(ParseException.class)
            .extracting(Throwable::getMessage)
            .asString()
            .isNotBlank();
    }
}
```

بعد از انجام این تست coverage برنامه به مقادیر زیر رسید.

JSONArray	100% (1/1)	11% (3/26)	2% (5/176)
JSONObject	100% (1/1)	30% (3/10)	19% (9/47)
JSONValue	100% (1/1)	50% (4/8)	8% (10/114)
JSONParser	100% (1/1)	73% (11/15)	42% (117/2...
ParseException	100% (1/1)	20% (2/10)	42% (12/28)
Yylex	100% (1/1)	72% (16/22)	70% (149/2...
Yytoken	100% (1/1)	100% (2/2)	66% (16/24)

Writing tests for valid strings

در این بخش ما توابع write را مورد آزمون قرار دادیم. بدین ترتیب همانطور که در تصویر می بینید بررسی کردیم که لیست‌ها، آبجکت‌های ساده، آبجکت‌های تو در تو مثل دیکشنری‌ها، مقادیر خالی، و تک مقادیرها به درستی تبدیل می‌شوند. دقت کنید که مقادیر این تست‌ها کاملاً مشابه تست اول است با این تفاوت که دیگر space در رشته‌ها نداریم. مشابه قسمت قبل از کلاس TestCase برای نوشتن حالات مختلف تست کمک گرفتیم.

```
List<TestCase> testCases = Arrays.asList(
    // simple arrays
    new TestCase( expected: "[1,2,30]", JSONArray.of(1L, 2L, 30L)),
    new TestCase( expected: "[\"a\", \"bbb\"]", JSONArray.of("a", "bbb")),
    new TestCase( expected: "[1.01,22.004]", JSONArray.of(1.01, 22.004)),











    // simple object
    new TestCase(
        expected: "{\"bb\":20,\"a\":false,\"ddd\":null}",
        JSONObject.of("bb", 20L, "a", false, "ddd", null)
    ),

    // nested object
    new TestCase(
        expected: "{\"aa\":[1.0,2.1,3.14],\"bb\":{\"ddd\":{\"t1\",\"t2\"},\"ccc\":\"text\"}}",
        JSONObject.of(
            "aa", JSONArray.of(1.0, 2.1, 3.14),
            "bb",
            JSONObject.of(
                "ccc", "text",
                "ddd", JSONArray.of("t1", "t2")
            )
        )
    ),

    // empty and null values
    new TestCase( expected: "[]", new JSONArray()),
    new TestCase( expected: "{}", new JSONObject()),
    new TestCase( expected: "null", given: null),

    // single values
    new TestCase( expected: "100", given: 100L),
    new TestCase( expected: "\"abc\"", given: "abc"),
    new TestCase( expected: "10.1", given: 10.1),
    new TestCase( expected: "false", given: false),
    new TestCase( expected: "true", given: true)
);
```

و در آخر بعد از افزودن این تست مقادیر coverage به مقادیر زیر رسید. همان طور که می بینید، coverage همه کلاس ها زیاد شده است.

Element —	Class, %	Method, %	Line, %
▼  org.json.simple	100% (7/7)	52% (49/93)	45% (364/799)
▼  parser	100% (4/4)	63% (31/49)	56% (286/502)
 ContainerFactory	100% (0/0)	100% (0/0)	100% (0/0)
 ContentHandler	100% (0/0)	100% (0/0)	100% (0/0)
 JSONParser	100% (1/1)	73% (11/15)	45% (114/251)
 ParseException	100% (1/1)	20% (2/10)	50% (11/22)
 Yylex	100% (1/1)	72% (16/22)	70% (146/206)
 Yytoken	100% (1/1)	100% (2/2)	65% (15/23)
 JSONArray	100% (1/1)	19% (5/26)	10% (17/162)
 JSONAware	100% (0/0)	100% (0/0)	100% (0/0)
 JSONObject	100% (1/1)	60% (6/10)	59% (25/42)
 JSONStreamAware	100% (0/0)	100% (0/0)	100% (0/0)
 JSONValue	100% (1/1)	87% (7/8)	38% (36/93)