



پر نامه سازی پیش رفته

آزمون نرم افزار

مهدی مصطفی زاده

سرفصل مطالب

- تضمین کیفیت نرمافزار (Software Quality Assurance)
- آزمون نرمافزار (Software Testing)
- آزمون واحد نرمافزار (Software Unit Testing)
- نوشتن آزمون واحد با JUnit
- بدلسازی در آزمون واحد (Mocking)





تضمين کيفيت نرم افزار

- ماريپندر، نخستين فضاپيمای پرئامهی فضایي ماريپندر ایالات متحده
- هزینه‌ی ۵/۸ ميليون دلار در سال ۱۹۶۲
- خروج از کنتل، مدت کوتاهی پس از پرتاب و انهدام به دستور افسر ایمنی (۵/۲۹۴ تانیه بعد)
- گفته می‌شود که علت قراموشی پرئامه‌نویس در قراردادن - پر روی \bar{R}_n در پخشی از پرئامه‌ی هدایت پوده است.



تضمين کيفيت نرم افزار (ادامه)

- دستگاه پرتو درمانی تراک - ۲۵ در کانادا به درستی کار نکرد و اشعه های کشنده ای را به بیماران تاپید. (۱۹۸۵)



- فوت چند بیمار

- به دلیل یک خطای کوچک نرم افزاری (شدايط مسابقه یا Race Condition) که باعث شد که دستگاه درست کار نماید.



آزمون نرم افزار

- آزمایش نرم افزار در شرایط مختلف (انواع ورودی‌ها، تحمیل میزان پار مختلف)
- به منظور افزایش کیفیت نرم افزار (کاهش احتمال وجود خطای در نرم افزار)
- انواع مختلف آزمون نرم افزار
- آزمون پذیرش، آزمون سیستم، آزمون تجمعی، آزمون پار، آزمون واحد و ...



آزمون واحد نرم افزار

- یکی از انواع آزمون نرم افزار که به آزمون واحدهای کوچک سازندهی نرم افزار (متدها) می پردازد و توسط محدوده نویس نوشتہ می شود.

- فراپند: ورودی های متدرآماده می کنیم.
- متدرآب ورودی های مذکور فراخوانی می کنیم و مُحروچی واقعی آن (actual value) را دریافت می کنیم.
- مُحروچی واقعی را با مُحروچی مورد انتظار (expected value) مقایسه می کنیم.
- این فراپند پاید به شکل محدود کار، تکرار پذیر و قطعی انجام شود.



نوشتن آزمون واحد با JUnit

- نوشتن آزمون واحد در چاوا، با استفاده از کتابخانه‌ی Junit امکان‌پذیر است.
- چون این کتابخانه، چند هسته‌ی اصلی چاوا نیست، پاید آن را به برنامه اضافه کنیم.
- کتابخانه‌های نوشته شده به زبان چاوا، در قالب فایل Jar عرضه می‌شوند.
- پاید فایل Jar مربوط به Junit را به مسیر درسترس پر نامه (classpath) اضافه کنیم.



افزودن Junit به برنامه

The screenshot shows the Maven repository search results for 'junit'. The page title is 'JUnit' and the subtitle is '2. JUnit'. It displays the following information:

- junit » junit**
- JUnit is a unit testing framework for Java, created by Erich Gamma and Kent Beck.
- Last Release on Jan 1, 2020
- Date**: (Jan 01, 2020)
- Files**: jar (372 KB) | View All
- Repositories**: Central
- 97,772 usages**
- EPL**

<https://mvnrepository.com/>

- ورود به سایت
- جستجوی عبارت Junit در پخش جستجوی بالای صفحه
- کلیک پر روی لینک مشخص شده

- انتخاب نسخه 4.13
- دانلود فایل Jar

Junit په کتابخانه hamcrest-core و اپته است. پس فایل Jar مربوط به نسخه 1.3 این کتابخانه را نیز دانلود نمایید.



افزودن JUnit به برنامه (ادامه)

- دو فایل Jar را در ریشه‌ی پروژه‌ی خود قرار دهید.
- هر دو فایل را انتخاب کرده، کلیک راست نمایید.
- از منوی Build Path گزینه‌ی Add to Build Path را انتخاب نمایید.



نوشتن اولین آزمون با JUnit

```
package math;

import static org.junit.Assert.*;
import org.junit.Test;

public class MathUtilTest {

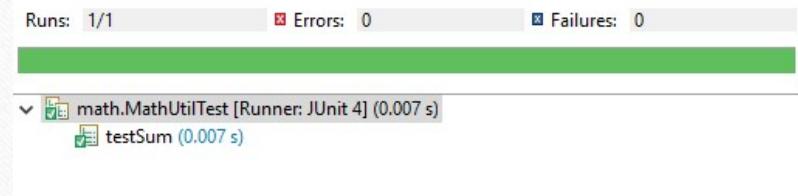
    MathUtil mathUtil = new MathUtil();

    @Test
    public void testSum() {
        int a = 10;
        int b = 2;
        int result = mathUtil.sum(a, b);
        assertEquals(12, result);
    }
}
```

```
package math;

public class MathUtil {

    int sum(int a, int b) {
        return a + b;
    }
}
```



آزمون پرتاب استثناء

```
int div(int a, int b) {  
    return a / b;  
}  
  
@Test  
public void testDiv_WHEN_secondParamIsNull_EXPECT_NPE() {  
    try {  
        a.div(10, 0);  
        fail();  
    } catch(NullPointerException e) {  
        assertTrue(true); // optional  
    }  
}
```

مشکل مورد آزمون زیر در چیست؟



مطالعه‌ی بیشتر

- Annotation های کمکی (@AfterClass، @BeforeClass، @After، @Before)
- الگوهای نام‌گذاری (آزمون‌های واحد در نقش مستندات)
- مدیریت و اپستگی‌ها (dependencies) با استفاده از ابزار Maven



بدل‌سازی در آزمون واحد (طرح مساله)

- می‌خواهیم متندی از یک کلاس را به روش آزمون واحد پیازماییم. متند و اپسته به متندها پی از کلاس‌های دیگر است. متندها پی که ...
- هنوز پیاده‌سازی نشده‌اند.
- پیاده‌سازی شده‌اند؛ اما راه‌اندازی آن‌ها ممکن نیست و یا دشوار است.
- واپستگی به پایگاه داده دارد.
- مریوط به کتابخانه‌ی زیرساختی هستند که هنوز انتخاب نشده است.



بدل سازی در آزمون واحد (مثال)

- آیا راهی پدای آزمون واحد fA بدون در اختیار داشتن پیاده سازی درست و کامل fb وجود دارد؟

```
public class A {  
    B b;  
    String fA(String str) {  
        return "Message: " + b.fb(str.toUpperCase());  
    }  
}
```

```
public class B {  
    String fb(String str) {  
        return null;  
    }  
}
```



بدل‌سازی در آزمون واحد (تحلیل مساله)

- پدای آزمون واحد، آیا اصلاً نیازی به واحدهای دیگر داریم؟
- مگر پدای پیاده‌سازی یک واحد، به پیاده‌سازی دیگر واحدها نیاز است؟
- تنها به فراخوانی درست و بجای آن واحد پا پارامتر مناسب و درست و استفاده‌ی صحیح از خروجی آن نیازمندیم.
- ۱. فراخوانی متدهای SALAM و رودی fb پا
- ۲. خروجی آن هرچه پاشد، رسنهی Message: "را به آن پیاقراپ و پرگرداند



بدلسازی در آزمون واحد (راهکار)

- تولید محوچی فرضی به ازای فراخوانی یک تابع در یک واحد، بدون آنکه آن تابع را اجرا کنیم.
- بدلسازی تنها تولید محوچی را در بد نمی گیرد؛ بلکه پیان انتظار ما از فراخوانی را نیز شامل می شود.
- بدلسازی = زوچ (فراخوانی، محوچی)
- مثال: زوچ (fb("SALAM"), "BYE")



بدل‌سازی با استفاده از JMockit

- نسخه‌ی 1.8 کتابخانه‌ی JMockit را دانلود کرده و به پرینامه‌ی خود اضافه نمایید.

```
package main;

import org.junit.runner.RunWith;
import mockit.Tested;
import mockit.integration.junit4.JMockit;

@RunWith(JMockit.class)
public class MockTest {

    @Tested
    A a;
}
```

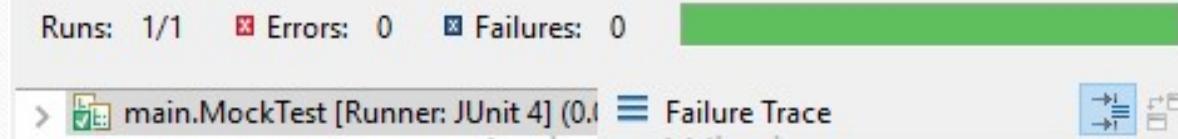


- سایر اپدراها:
 - Mockito
 - EasyMock
 - ...



تعريف اشیاء بدلی برای وابستگی‌ها

```
public class A {  
    B b;  
  
    String fA(String str) {  
        return "Message: " + b.fb(str.toUpperCase());  
    }  
}  
  
@Tested  
A a;  
  
@Injectable  
B b;  
  
@Test  
public void testIsNotNull() {  
    assertNotNull(a.b);  
}
```



بدل‌سازی ساده با یک فراخوانی

```
@Test
public void testIsNotNull() {
    new Expectations() {
        {
            b.fb("SALAM"); result = "BYE";
        }
    };

    String result = a.fA("Salam");
    assertEquals("Message: BYE", result);
}
```

- می‌خواهیم متده آزمون‌شونده (fA) را با ورودی "Salam" بیازماییم.
- انتظار این است که: متده fb با ورودی "SALAM" فراخوانی شود.
- خروجی متده fb هرچه پاشد، رشته‌ی "Message: BYE" بپذیرد.



بدل‌سازی با چند فراخوانی

```
public class A {  
  
    B b;  
  
    String fA(String str) {  
        return "Message: " + b.fb(str.toUpperCase()) + @Test  
            " (" + b.fb(str.toLowerCase()) + ")";  
    }  
}  
  
    public void testIsNotNull() {  
        new Expectations() {  
            {  
                b.fb("SALAM"); result = "BYE";  
                b.fb("salam"); result = "bye";  
            }  
        };  
  
        String result = a.fA("Salam");  
        assertEquals("Message: BYE (bye)", result);  
    }  
}
```



بدل‌سازی با دفعات فرآخوانی و محدودیت‌های مختلف

...، maxTimes، minTimes، times، withInstanceOf، anyDouble، anyLong، anyInt، anyString

```
@Test
public void testIsNotNull() {
    new Expectations() {
        {
            b.fb(anyString); result = "BYE"; times = 1;
            b.fb(withInstanceOf(String.class)); result = "bye"; minTimes = 1;
        }
    };

    String result = a.fA("Salam");
    assertEquals("Message: BYE (bye)", result);
}
```



استثناء به عنوان خروجی بدلی

```
String fA(String str) {
    try {
        return "Message: " + b.fb(null);
    } catch(Exception e) {
        return e.getMessage();
    }
}                                     @Test
                                         public void testIsNotNull() {
                                         new Expectations() {
                                         {
                                             b.fb(null); result = new RuntimeException("null arg");
                                         }
                                         };
                                         String result = a.fA("Salam");
                                         assertEquals("null arg", result);
                                         }
```



بدل‌سازی پارامتر ورودی

```
public class A {
    String fA(String str, B b) {
        try {
            return "Message: " + b.fb(null);
        } catch(Exception e) {
            return e.getMessage();
        }
    }
}

@ Injectable
B b;

@Test
public void testIsNotNull() {
    new Expectations() {
        {
            b.fb(null);
            result = new RuntimeException("null arg");
        }
    };
    String result = a.fA("Salam", b);
    assertEquals("null arg", result);
}
```



بدل سازی اشیاء نمونه سازی شده

```
A a;  
@Mocked  
B b;  
  
@Test  
public void testIsNotNull() {  
    new NonStrictExpectations() {  
        {  
            b.fb("SALAM");  
            result = "BYE";  
        }  
    };  
    String result = a.fA("Salam");  
    assertEquals("Message: BYE", result);  
}
```

```
public class A {  
    String fA(String str) {  
        B b = new B();  
        return "Message: " + b.fb(str.toUpperCase());  
    }  
}
```

- هنگام استفاده از `@Mocked`، تمامی اشیاء `new` شده از کلاس مورد نظر، بدل می شوند (اسم شیء مهم نیست).



پیاده‌سازی بدلی

```
@Test
public void test() {
    new MockUp<B>() {
        @Mock
        String fb(String str) {
            if(str.equals("SALAM"))
                return "BYE";
            return "Good Bye!";
        }
    };

    String result = a.fA("Salam");
    assertEquals("Message: BYE", result);
}
```

- گاهی لازم است منطق متدی که متد آزمون‌شونده به آن وابسته است را تغییر دهیم.
- به غیر از متدهای عادی، می‌توان سازنده، متدهای ایستا و حتی واسطها را نیز بدل کرد.
- برای پیاده‌سازی بدلی سازنده، متدی با عنوان \$init و نوع خروجی void تعریف می‌کنیم.
- منطق سایر متدها، به شکل عادی اجرا می‌شود.



بدل‌سازی جزئی

```
@Tested
A a;

@Test
public void testIsNotNull() {
    new Expectations(a) {
        {
            a.fb("SALAM");
            result = "BYE";
        }
    };
    String result = a.fA("Salam");
    assertEquals("Message: BYE", result);
}
```

```
public class A {

    String fA(String str) {
        return "Message: " + fb(str.toUpperCase());
    }

    String fb(String upperCase) {
        return null;
    }
}
```

