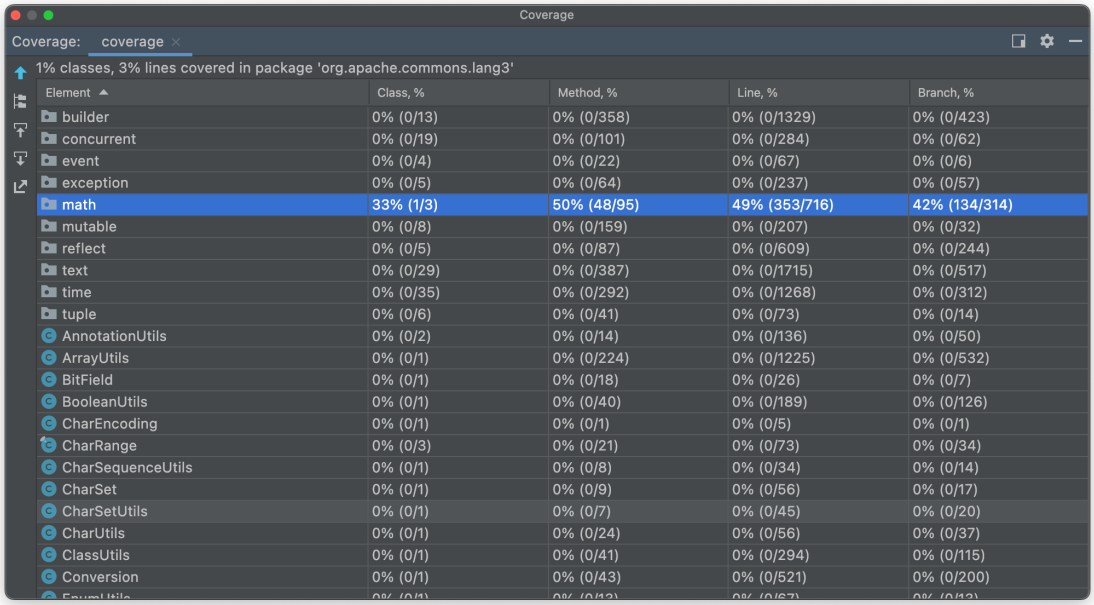
2022_Spring Software-Engineering

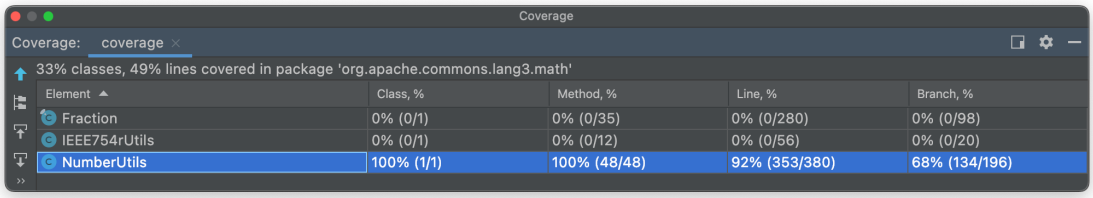
Assignment #1

20191128 Jian Park

[Lang_7]

- **Buggy Class :** `org.apache.commons.lang3.math.NumberUtils`
- **Branch Coverage :** 80.87855297157622% (in statistics.csv)





We created and executed a test suite specifically for `math/NumberUtils`. Looking at the picture above, we can see that coverage of the `NumberUtils` class is much higher than that of other classes. Nevertheless, it is actually not a very high value of 68%.

- **Bug :**

```
445 445     public static Number createNumber(String str) throws NumberFormatException {
446 446         if (str == null) {
447 447             return null;
448 448         }
449 449         if (StringUtils.isBlank(str)) {
450 450             throw new NumberFormatException("A blank string is not a valid number");
451 451         }
452         if (str.startsWith("--")) {
453             return null;
454         }
```

```
713 710     public static BigDecimal createBigDecimal(String str) {
714 711         if (str == null) {
715 712             return null;
716 713         }
717 714         // handle JDK1.3.1 bug where "" throws IndexOutOfBoundsException
718 715         if (StringUtils.isBlank(str)) {
719 716             throw new NumberFormatException("A blank string is not a valid number");
720 717         }
    718         if (str.trim().startsWith("--")) {
721 719             // this is protection for poorness in java.lang.BigDecimal.
722 720             // it accepts this as a legal value, but it does not appear
723 721             // to be in specification of class. OS X Java parses it to
724 722             // a wrong value.
    723             throw new NumberFormatException(str + " is not a valid number.");
    724         }
725 725         return new BigDecimal(str);
726 726     }
```

✓ Red line is buggy version, green line is fixed version

The `createBigDecimal` method is a method that converts `String` into a `BigDecimal` number and returns it. The `CreateNumber` checks that given string start with '--', and returns null if found. This is a solution for a bug in `BigDecimal` that the string starting with '--' is not a number. But it should be throw exception likes other mothods. And also, should be move to `createBigDecimal` for checked all of the `BigDecimal` case.

➘ **Bug-revealing test :** Tests run: 146, Failures: 2, Errors: 0, Skipped: 0

1. NumberUtils_ESTest.test106

```
➜ java.lang.AssertionError: Exception was not thrown in org.apache.commons.lang3.math.NumberUtils but i
n java.math.BigDecimal.<init>(BigDecimal.java:553): java.lang.NumberFormatException
    at org.evosuite.runtime.EvoAssertions.assertThrownBy(EvoAssertions.java:112)
    at org.evosuite.runtime.EvoAssertions.verifyException(EvoAssertions.java:49)
    at org.apache.commons.lang3.math.NumberUtils_ESTest.test106(NumberUtils_ESTest.java:1904)
```

```
@Test(timeout = 4000)
public void test106()  throws Throwable  {
    float[] floatArray0 = new float[5];
    floatArray0[0] = (-387.2784F);
    floatArray0[1] = 271.4187F;
    floatArray0[2] = (-4066.772F);
    floatArray0[3] = 1.0F;
    floatArray0[4] = 0.0F;
    NumberUtils.max(floatArray0);
    NumberUtils.min(0.0F, 645.99F, 271.4187F);
    double[] doubleArray0 = new double[7];
    doubleArray0[0] = (double) 271.4187F;
    doubleArray0[1] = (-1202.92496144487);
    doubleArray0[2] = (double) 0.0F;
    doubleArray0[3] = (double) (-4066.772F);
    doubleArray0[4] = (double) (-387.2784F);
    doubleArray0[5] = 0.0;
    doubleArray0[6] = (double) 271.4187F;
    NumberUtils.min(doubleArray0);
    NumberUtils.toLong("-0XThe Array must not be null");
    NumberUtils.isNumber("---0XrB□mFL<%*%YC0Dv|k\"");
    NumberUtils.toShort("-0XThe Array must not be null");
    // Undeclared exception!
    try {
      NumberUtils.createBigDecimal("---0XrB□mFL<%*%YC0Dv|k\"");
      fail("Expecting exception: NumberFormatException");

    } catch(NumberFormatException e) {
       //
       // ---0XrB□mFL<%*%YC0Dv|k\" is not a valid number.
       //
       verifyException("org.apache.commons.lang3.math.NumberUtils", e);
    }
}
```

We wanted to test the functions of the `NumberUtils` class using the various types of

numbers, and to confirm that transfer the invalid string value (start with '--') to a number would be throw `NumberFormatException` in `createBigDecimal`. But in this code, exception was thrown in basic class of java, not in `createBigDecimal` of Lang7. Because `createBigDecimal` method doesn't have any handle of this invalid string value case.

2. NumberUtils_ESTest.test142

```
➜ java.lang.AssertionError: Expecting exception: NumberFormatException
    at org.apache.commons.lang3.math.NumberUtils_ESTest.test142(NumberUtils_ESTest.java:2624)
```

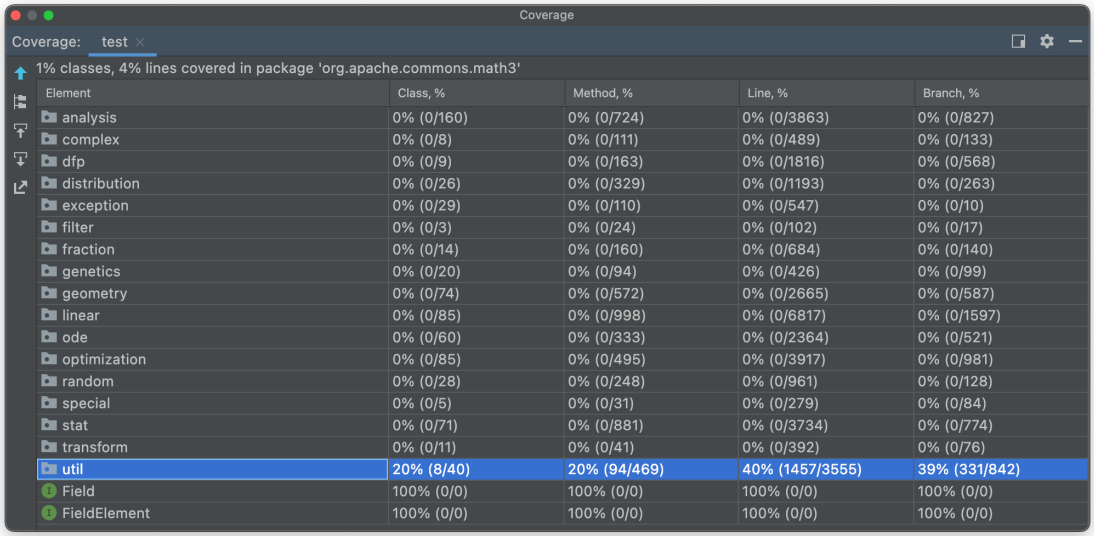
```
@Test(timeout = 4000)
public void test142()  throws Throwable  {
    short[] shortArray0 = new short[4];
    shortArray0[0] = (short) (-9334);
    shortArray0[1] = (short)48;
    shortArray0[2] = (short)1869;
    shortArray0[3] = (short) (-2239);
    NumberUtils.max(shortArray0);
    int[] intArray0 = new int[1];
    intArray0[0] = (int) (short)1869;
    NumberUtils.min(intArray0);
    int[] intArray1 = new int[4];
    intArray1[0] = (int) (short) (-9334);
    intArray1[1] = (int) (short) (-9334);
    intArray1[2] = (int) (short) (-9334);
    intArray1[3] = (int) (short)1869;
    NumberUtils.max(intArray1);
    String string0 = "---";
    NumberUtils.toInt("---");
    byte[] byteArray0 = new byte[3];
    byteArray0[0] = (byte)64;
    byteArray0[2] = (byte)16;
    NumberUtils.max(byteArray0);
    NumberUtils.min(shortArray0);
    try {
      NumberUtils.createNumber("---");
      fail("Expecting exception: NumberFormatException");

    } catch(NumberFormatException e) {
       //
       // --- is not a valid number.
       //
       verifyException("org.apache.commons.lang3.math.NumberUtils", e);
    }
}
```

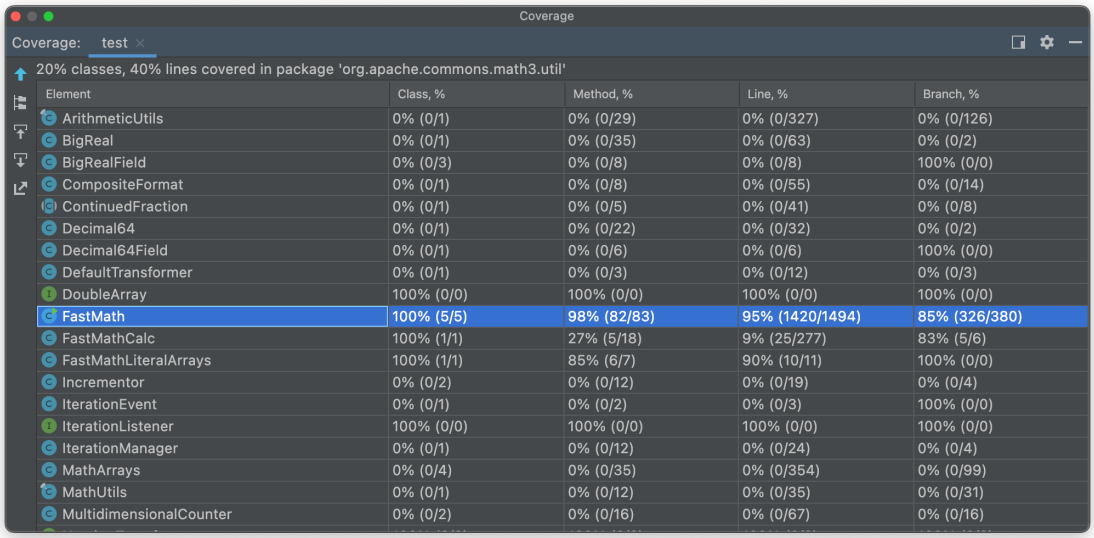
In this test, given string is start with '--'. So in `CreateNumber` method, it will be return null. And it makes `NumberFormatException`.

[Math_15]

➘ **Buggy Class :** `org.apache.commons.math3.util.FastMath`

➘ **Branch Coverage :** 87.22891566265061% ➜ 87.46987951807229% (after modify)





We created and executed a test suite specifically for `util/FastMath`. Looking at the picture above, we can see that coverage of the `FastMath` class is much higher than that of other classes. And it has a high value of 85%.

➘ **Bug :**

```
 312  312    /** 2^53 - double numbers this large must be even. */
      313    private static final double TWO_POWER_53 = 2 * TWO_POWER_52;
1538 1539        /* Handle special case x<0 */
1539 1540        if (x < 0) {
1540 1541            // y is an even integer in this case
1541              -     if (y >= TWO_POWER_52 || y <= -TWO_POWER_52) {
      1542        +     if (y >= TWO_POWER_53 || y <= -TWO_POWER_53) {
1542 1543                return pow(-x, y);
1543 1544            }
1544 1545
1545 1546            if (y == (long) y) {
1546 1547                // If y is an integer
1547 1548                return ((long)y & 1) == 0 ? pow(-x, y) : -pow(-x, y);
1548 1549            } else {
1549 1550                return Double.NaN;
1550 1551            }
1551 1552        }
```

✓ Red line is buggy version, green line is fixed version

The `Pow` is a method that returns the result of a power function with a given double number `x` is base and double number `y` is exponent. In IEEE 754, the frac of double precision is 52 bits. So, range of double precision is $[-2^{53}, +2^{53}]$ because frac has one free bit (higher one true bit doesn't present). If it's out of range, it will be rounded to a multiple of two powers (ex. $(2^{53}, 2^{54}] \rightarrow 2^{54}$) and it is even number. We want to handle it differently depending on whether `y` is an even integer or odd integer when `x` is negative. Therefore, the line 1541 should be `TWO_POWER_53`, not `TWO_POWER_52`. In this code, if y is odd integer in range $[2^{52}, 2^{53})$, it will return invalid value that `pow(-x, y)` instead of `-pow(x, y)`. And if y is neither integer nor special value in range $[2^{52}, 2^{53})$, it will also return invalid value that `pow(-x, y)` instead of `Double.NaN`.

➘ **Bug-revealing test :** Fail to generate a bug-revealing test. So I modify one of the generated test. As I mentioned before, if exponent is odd integer in range $[2^{52}, 2^{53})$ when base is negative, it will return invalid value that `pow(-x, y)` instead of `-pow(x, y)`. So, I add that case.

1. FastMath_ESTest.test131 (modify!)

```
→ java.lang.AssertionError: expected:<-Infinity> but was:<Infinity>
      at org.junit.Assert.fail(Assert.java:88)
      at org.junit.Assert.failNotEquals(Assert.java:834)
      at org.junit.Assert.assertEquals(Assert.java:553)
      at org.junit.Assert.assertEquals(Assert.java:683)
      at org.apache.commons.math3.util.FastMath_ESTest.test131(FastMath_ESTest.java:12441)
```

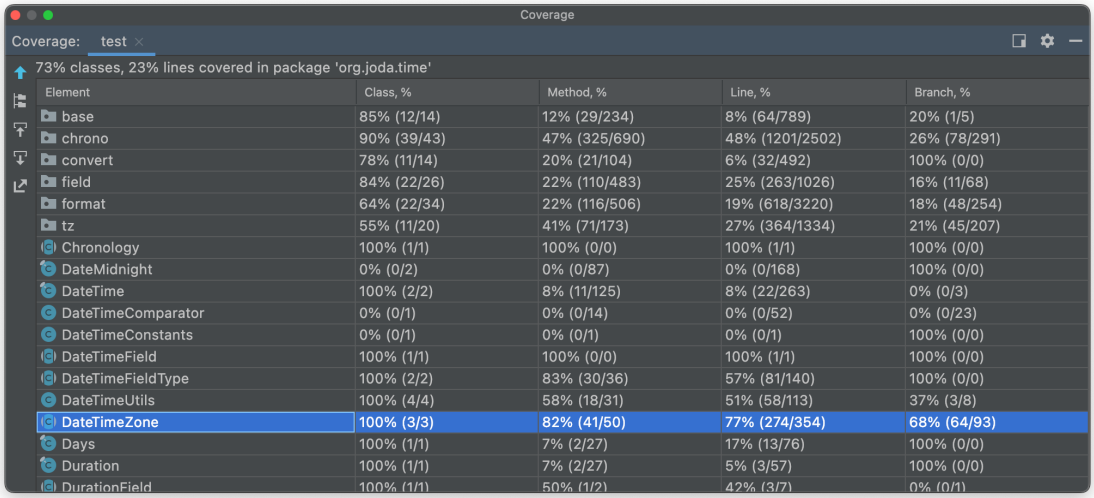
```
@Test(timeout = 4000)
public void test131()  throws Throwable {
  double double0 = FastMath.pow(-3.0, 4503599627370497.0);
  double double1 = FastMath.pow(-3.0, 4503599627370495.0);
  assertEquals(double1 * 9, double0, 0.01);
}
```

In this test, both `double0` and `double1` bases are `-3` that negative number. Exponent of `double0` is `4503599627370497.0` $(= (-3)^{2^{52}+1} = (-3)^2 * (-3)^{2^{52}-1})$ that odd integer in range $[2^{52}, 2^{53})$. We expected the result to be same as the `double1` which has exponent is `4503599627370495.0` $(= (-3)^{2^{52}-1})$. But exponent of `double0` in range $[2^{52}, 2^{53})$. So it goes into the if statement in line 1541 and return not corret value `pow(-x, y)`. And double1 is return correct value `-pow(x, y)`.

[Time_8]

➘ **Buggy Class :** `org.joda.time.DateTimeZone`

➘ **Branch Coverage :** 80.48780487804879% (in statistics.csv)



Element	Class, %	Method, %	Line, %	Branch, %
base	85% (12/14)	12% (29/234)	8% (64/789)	20% (1/5)
chrono	90% (39/43)	47% (325/690)	48% (1201/2502)	26% (78/291)
convert	78% (11/14)	20% (21/104)	6% (32/492)	100% (0/0)
field	84% (22/26)	22% (110/483)	25% (263/1026)	16% (11/68)
format	64% (22/34)	22% (116/506)	19% (618/3220)	18% (48/254)
tz	55% (11/20)	41% (71/173)	27% (364/1334)	21% (45/207)
Chronology	100% (1/1)	100% (0/0)	100% (1/1)	100% (0/0)
DateMidnight	0% (0/2)	0% (0/87)	0% (0/168)	100% (0/0)
DateTime	100% (2/2)	8% (11/125)	8% (22/263)	0% (0/3)
DateTimeComparator	0% (0/1)	0% (0/14)	0% (0/52)	0% (0/23)
DateTimeConstants	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
DateTimeField	100% (1/1)	100% (0/0)	100% (1/1)	100% (0/0)
DateTimeFieldType	100% (2/2)	83% (30/36)	57% (81/140)	100% (0/0)
DateTimeUtils	100% (4/4)	58% (18/31)	51% (58/113)	37% (3/8)
DateTimeZone	100% (3/3)	82% (41/50)	77% (274/354)	68% (64/93)
Days	100% (1/1)	7% (2/27)	17% (13/76)	100% (0/0)
Duration	100% (1/1)	7% (2/27)	5% (3/57)	100% (0/0)
DurationField	100% (1/1)	50% (1/2)	42% (3/7)	0% (0/1)

We created and executed a test suite specifically for `util/FastMath`. Looking at the picture above, we can see that coverage of the `FastMath` class is much higher than that of other classes. Nevertheless, it is actually not a very high value of 68%.

➘ **Bug :**

```
272 272     public static DateTimeZone forOffsetHoursMinutes(int hoursOffset, int minutesOffset) throws IllegalArgumentException {
273 273         if (hoursOffset == 0 && minutesOffset == 0) {
274 274             return DateTimeZone.UTC;
275 275         }
276 276         if (hoursOffset < -23 || hoursOffset > 23) {
277 277             throw new IllegalArgumentException("Hours out of range: " + hoursOffset);
278 278         }
279         -   if (minutesOffset < 0 || minutesOffset > 59) {
    279     +   if (minutesOffset < -59 || minutesOffset > 59) {
280 280             throw new IllegalArgumentException("Minutes out of range: " + minutesOffset);
281 281         }
    282     +   if (hoursOffset > 0 && minutesOffset < 0) {
    283     +       throw new IllegalArgumentException("Positive hours must not have negative minutes: " + minutesOffset);
    284     +   }
282 285         int offset = 0;
283 286         try {
284 287             int hoursInMinutes = hoursOffset * 60;
285 288             if (hoursInMinutes < 0) {
286         -               minutesOffset = hoursInMinutes - minutesOffset;
    289     +               minutesOffset = hoursInMinutes - Math.abs(minutesOffset);
287 290             } else {
288 291                 minutesOffset = hoursInMinutes + minutesOffset;
289 292             }
290 293             offset = FieldUtils.safeMultiply(minutesOffset, DateTimeConstants.MILLIS_PER_MINUTE);
291 294         } catch (ArithmeticException ex) {
292 295             throw new IllegalArgumentException("Offset is too large");
293 296         }
294 297         return forOffsetMillis(offset);
295 298     }
```

✓ Red line is buggy version, green line is fixed version

The `forOffsetHourMinutes` method is gets a time zone instance for the specified offset to UTC. In UTC notation, negative(-) means slower than standard time. And of course, minutes can be negative when hour is negative. When hour is negative, it means that time is as slow as the sum of the absolute value of hour and minute. ex) -15h, -30m → It's 15:30 slower. -15h, 30m → It's 15:30 slower too. So, even if the minute is negative, it should be allowed. And we should be handled the case that both hour and minute are negative. At now, `forOffsetHourMinutes` method should

have thrown a `IllegalArgumentException` when `minutesOffset` is negative.

➘ **Bug-revealing test :** Tests run: 81, Failures: 2, Errors: 2, Skipped: 0

1. test56(org.joda.time.DateTimeZone_ESTest)

```
➜ org.evosuite.runtime.mock.java.lang.MockIllegalArgumentException: Minutes out of range: -1
        at org.joda.time.DateTimeZone.forOffsetHoursMinutes(DateTimeZone.java:280)
        at org.joda.time.DateTimeZone_ESTest.test56(DateTimeZone_ESTest.java:3269)
```

```java
@Test(timeout = 4000)
public void test56()  throws Throwable  {
    DateTimeZone dateTimeZone0 = DateTimeZone.forOffsetHoursMinutes((-1), (-1));
    assertNotNull(dateTimeZone0);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());

    int int0 = dateTimeZone0.UTC.getOffsetFromLocal((-1));
    assertEquals(0, int0);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());

    TimeZone timeZone0 = dateTimeZone0.toTimeZone();
    assertNotNull(timeZone0);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("GMT-01:01", timeZone0.getID());

    long long0 = dateTimeZone0.UTC.convertUTCToLocal(100L);
    assertEquals(100L, long0);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());

    boolean boolean0 = dateTimeZone0.isFixed();
    assertTrue(boolean0);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());

    DateTimeZone dateTimeZone1 = DateTimeZone.forTimeZone(timeZone0);
    assertNotNull(dateTimeZone1);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("GMT-01:01", timeZone0.getID());
    assertTrue(dateTimeZone1.isFixed());
    assertEquals("-01:01", dateTimeZone1.getID());
    assertEquals("-01:01", dateTimeZone1.toString());
    assertSame(dateTimeZone0, dateTimeZone1);
    assertSame(dateTimeZone1, dateTimeZone0);

    String string0 = dateTimeZone0.getNameKey(0L);
    assertNull(string0);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());
    assertSame(dateTimeZone0, dateTimeZone1);

    long long1 = dateTimeZone0.adjustOffset((-2153L), false);
    assertEquals((-2153L), long1);
    assertEquals("-01:01", dateTimeZone0.getID());
    assertEquals("-01:01", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());
    assertSame(dateTimeZone0, dateTimeZone1);
    assertFalse(long1 == long0);

    LinkedList<Locale.LanguageRange> linkedList0 = new LinkedList<Locale.LanguageRange>();
    assertNotNull(linkedList0);
    assertEquals(0, linkedList0.size());

    LinkedList<Locale.LanguageRange> linkedList1 = new LinkedList<Locale.LanguageRange>();
    assertNotNull(linkedList1);
    assertEquals(0, linkedList1.size());
```

```
    assertTrue(linkedList1.equals((Object)linkedList0));
}
```

In this test, we expected to create a `DateTimeZone` with a value of "-01:01" and apply several functions to it and verify it's correctness. But in the highlighted line, the `minutesOffset` is negative (-1). So, it goes into the if statement in line 279 of the `forOffsetHourMinutes` method, and throws an `IllegalArgumentException`.

2. test59(org.joda.time.DateTimeZone_ESTest)

```
➜ org.evosuite.runtime.mock.java.lang.MockIllegalArgumentException: Minutes out of range: -23
        at org.joda.time.DateTimeZone.forOffsetHoursMinutes(DateTimeZone.java:280)
        at org.joda.time.DateTimeZone_ESTest.test59(DateTimeZone_ESTest.java:3466)
```

```
@Test(timeout = 4000)
public void test59()  throws Throwable  {
    BuddhistChronology buddhistChronology0 = BuddhistChronology.getInstanceUTC();
    assertEquals(1, BuddhistChronology.BE);
    assertNotNull(buddhistChronology0);

    DateTimeZone dateTimeZone0 = DateTimeZone.forOffsetHoursMinutes((-23), (-23));
    assertNotNull(dateTimeZone0);
    assertEquals("-23:23", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("-23:23", dateTimeZone0.getID());
}
```

In this test, we expected to create a `DateTimeZone` with a value of "-23:23" and apply several functions to it and verify it's correctness. But in the highlighted line, the `minutesOffset` is negative (-23). So, it goes into the if statement in line 279 of the `forOffsetHourMinutes` method, and throws an `IllegalArgumentException`.

Conclusion

I have created many programs, but this was the first time I had used a test generator to check it's correctness. It is difficult to test as the code gets longer and more complicated, but using EvoSuite, it was convenient to generate a test with a simple command in the direction of automatically increasing branch coverage. And using EvoSuite, I found some interesting things.

In the Math8 package, ran a test on `FastMath`, a class for fast and accurate calculations. and I found that variables with values appeared in other test suites.

```
@Test(timeout = 4000)
public void test000()  throws Throwable  {
    double double0 = FastMath.floor(3.4893601256685762E283);
    assertEquals(3.4893601256685762E283, double0, 0.01);

    double double1 = FastMath.exp(3.7072473866919033E-183);
    assertNotEquals(double1, double0, 0.01);
    assertEquals(1.0, double1, 0.01);

    double double2 = FastMath.sqrt(3.7072473866919033E-183);
    assertNotEquals(double2, double1, 0.01);
    assertNotEquals(double2, double0, 0.01);
    assertEquals(6.088716931088112E-92, double2, 0.01);

    double double3 = FastMath.abs(0.07419405760538333);
    assertNotEquals(double3, double0, 0.01);
    assertNotEquals(double3, double1, 0.01);
    assertNotEquals(double3, double2, 0.01);
    assertEquals(0.07419405760538333, double3, 0.01);

    float float0 = FastMath.nextUp(1623.06F);
    assertEquals(1623.0602F, float0, 0.01F);
 …
}
```

```
@Test(timeout = 4000)
public void test130()  throws Throwable  {
    FastMath.floor(3.4893601256685762E283);
    FastMath.exp(3.7072473866919033E-183);
    FastMath.sqrt(3.7072473866919033E-183);
    FastMath.abs(3.7072473866919033E-183);
    float float0 = FastMath.nextUp(1623.06F);
    assertEquals(1623.0602F, float0, 0.01F);

    double double0 = FastMath.sin(1623.06F);
    assertEquals(0.9100245640753561, double0, 0.01);

    float float1 = FastMath.min((-2933.0F), 1623.0602F);
 …
}
```

The above is the code of `test000` and `test130`. If you look closely at the code, you can see that three numbers `3.4893601256685762E283`, `3.70724738669190 33E-183`, and `1623.0602F` overlap despite different tests. I think this is a characteristic of EvoSuite using the evolution generate method that mutates the randomly generated population at the beginning. In EvoSuite, test suites are heavily influenced by initial population. In the process of selecting the seed test and mutating the selected tests using various mutation operators such as AOR and ROR COR, the test was created while some of the values were maintained.

However, there were cases in Math15 where the bug could not be found even if the best test suite was generated after several mutates. I think this is because EvoSuite is a Whitebox test. In other words, since it is a structure-based test, the purpose is only to increase coverage. Therefore, it is not interested in whether all functions operate normally. Just terminating the test generation as soon as the coverage goal is exceeded. Actually, Class, Method, and Line coverage is over then 95%, and also branch coverage is 87% but it failed to find bug.

Element	Class, %	Method, %	Line, %
FastMath	100% (5/5)	98% (82/83)	95% (1420/1494)

In addition, if the bug is very unlikely (with a small range of bugs), no matter how random the test is, it may not be possible to generate a test that finds bugs. A bug in the `Math15` class occurs only when a given argument is in a specific range that $[2^{52}, 2^{53})$, which occurs in the `pow(x, y)` method. Considering the range of numbers that can be expressed in 64 bits, and probability of using the buggy class method in the test, the probability of finding a bug is even lower., it can be seen that this is very unlikely to occur. Through this example, we learned that simply increasing the coverage of the test does not necessarily mean that it is effective.

Although the test suite was created in the fixed `Time8` package, 2 tests were failed regardless of whether they were fixed or buggy versions. In both tests, the `DataTimeZone` provider was set to `UTCProvider`, but it was not set to `UTC`. I wanted to find out the cause of this, but I have not yet revealed it. Below is the failed test code and message.

1. test01(org.joda.time.DateTimeZone_ESTest)

```
➜ org.junit.ComparisonFailure: expected:<[UTC]> but was:<[Asia/Seoul]>
        at org.junit.Assert.assertEquals(Assert.java:115)
        at org.junit.Assert.assertEquals(Assert.java:144)
        at org.joda.time.DateTimeZone_ESTest.test01(DateTimeZone_ESTest.java:103)
```

```java
@Test(timeout = 4000)
public void test01()  throws Throwable  {
    boolean boolean0 = FileSystemHandling.shouldThrowIOException((EvoSuiteFile) null);
    assertFalse(boolean0);

    UTCProvider uTCProvider0 = new UTCProvider();
    assertNotNull(uTCProvider0);

    Set<String> set0 = uTCProvider0.getAvailableIDs();
    assertNotNull(set0);
    assertEquals(1, set0.size());
    assertFalse(set0.isEmpty());

    Set<String> set1 = uTCProvider0.getAvailableIDs();
    assertNotNull(set1);
    assertEquals(1, set1.size());
    assertFalse(set1.isEmpty());
    assertNotSame(set1, set0);
    assertTrue(set1.equals((Object)set0));
```

```
    DateTimeZone.setProvider(uTCProvider0);
    DateTimeZone dateTimeZone0 = DateTimeZone.getDefault();
    assertNotNull(dateTimeZone0);
    assertEquals("UTC", dateTimeZone0.getID());
    assertEquals("UTC", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());

    Locale locale0 = Locale.GERMAN;
    assertNotNull(locale0);
    assertEquals("de", locale0.toString());
    assertEquals("deu", locale0.getISO3Language());
    assertEquals("de", locale0.getLanguage());
    assertEquals("", locale0.getCountry());
    assertEquals("", locale0.getISO3Country());
    assertEquals("", locale0.getVariant());

    DateTimeUtils.setCurrentMillisSystem();
    String string0 = dateTimeZone0.getName(106109248L);
    assertEquals("+00:00", string0);
    assertNotNull(string0);
    assertEquals("UTC", dateTimeZone0.getID());
    assertEquals("UTC", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());

    String string1 = dateTimeZone0.getID();
    assertEquals("UTC", string1);
    assertNotNull(string1);
    assertEquals("UTC", dateTimeZone0.getID());
    assertEquals("UTC", dateTimeZone0.toString());
    assertTrue(dateTimeZone0.isFixed());
    assertFalse(string1.equals((Object)string0));

    DateTimeZone dateTimeZone1 = DateTimeZone.forOffsetMillis(662);
    assertNotNull(dateTimeZone1);
    assertEquals("+00:00:00.662", dateTimeZone1.getID());
    assertEquals("+00:00:00.662", dateTimeZone1.toString());
    assertTrue(dateTimeZone1.isFixed());
    assertNotSame(dateTimeZone1, dateTimeZone0);
    assertFalse(dateTimeZone1.equals((Object)dateTimeZone0));

    boolean boolean1 = dateTimeZone1.isStandardOffset((-2752832L));
    assertTrue(boolean1);
    assertEquals("+00:00:00.662", dateTimeZone1.getID());
    assertEquals("+00:00:00.662", dateTimeZone1.toString());
    assertTrue(dateTimeZone1.isFixed());
    assertNotSame(dateTimeZone1, dateTimeZone0);
    assertFalse(dateTimeZone1.equals((Object)dateTimeZone0));
    assertFalse(boolean1 == boolean0);
}
```

2. test78(org.joda.time.DateTimeZone_ESTest)

```
➜ java.lang.AssertionError: null
        at org.junit.Assert.fail(Assert.java:86)
        at org.junit.Assert.assertTrue(Assert.java:41)
        at org.junit.Assert.assertTrue(Assert.java:52)
        at org.joda.time.DateTimeZone_ESTest.test78(DateTimeZone_ESTest.java:5052)
```

```
@Test(timeout = 4000)
public void test78()  throws Throwable  {
    UTCProvider uTCProvider0 = new UTCProvider();
    assertNotNull(uTCProvider0);

    Set<String> set0 = uTCProvider0.getAvailableIDs();
    assertNotNull(set0);
    assertEquals(1, set0.size());
    assertFalse(set0.isEmpty());

    Set<String> set1 = uTCProvider0.getAvailableIDs();
    assertNotNull(set1);
    assertEquals(1, set1.size());
    assertFalse(set1.isEmpty());
    assertNotSame(set1, set0);
    assertTrue(set1.equals((Object)set0));

    DateTimeZone.setProvider(uTCProvider0);
    DateTimeZone dateTimeZone0 = DateTimeZone.getDefault();
```

```
    assertNotNull(dateTimeZone0);
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("UTC", dateTimeZone0.toString());
    assertEquals("UTC", dateTimeZone0.getID());

    Locale locale0 = Locale.GERMAN;
    assertNotNull(locale0);
    assertEquals("deu", locale0.getISO3Language());
    assertEquals("", locale0.getCountry());
    assertEquals("de", locale0.toString());
    assertEquals("", locale0.getVariant());
    assertEquals("de", locale0.getLanguage());
    assertEquals("", locale0.getISO3Country());

    DateTimeUtils.setCurrentMillisSystem();
    String string0 = dateTimeZone0.getName(106109248L);
    assertEquals("+00:00", string0);
    assertNotNull(string0);
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("UTC", dateTimeZone0.toString());
    assertEquals("UTC", dateTimeZone0.getID());

    String string1 = dateTimeZone0.getID();
    assertEquals("UTC", string1);
    assertNotNull(string1);
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("UTC", dateTimeZone0.toString());
    assertEquals("UTC", dateTimeZone0.getID());
    assertFalse(string1.equals((Object)string0));

    DateTimeZone dateTimeZone1 = DateTimeZone.forOffsetMillis(662);
    assertNotNull(dateTimeZone1);
    assertEquals("+00:00:00.662", dateTimeZone1.toString());
    assertTrue(dateTimeZone1.isFixed());
    assertEquals("+00:00:00.662", dateTimeZone1.getID());
    assertNotSame(dateTimeZone1, dateTimeZone0);
    assertFalse(dateTimeZone1.equals((Object)dateTimeZone0));

    boolean boolean0 = dateTimeZone1.isStandardOffset((-2752832L));
    assertTrue(boolean0);
    assertEquals("+00:00:00.662", dateTimeZone1.toString());
    assertTrue(dateTimeZone1.isFixed());
    assertEquals("+00:00:00.662", dateTimeZone1.getID());
    assertNotSame(dateTimeZone1, dateTimeZone0);
    assertFalse(dateTimeZone1.equals((Object)dateTimeZone0));

    Set<String> set2 = DateTimeZone.getAvailableIDs();
    assertNotNull(set2);
    assertEquals(1, set2.size());
    assertFalse(set2.isEmpty());
    assertTrue(set2.contains(string1));
    assertFalse(set2.contains(string0));

    DateTimeZone dateTimeZone2 = DateTimeZone.forOffsetMillis(662);
    assertNotNull(dateTimeZone2);
    assertEquals("+00:00:00.662", dateTimeZone2.toString());
    assertTrue(dateTimeZone2.isFixed());
    assertEquals("+00:00:00.662", dateTimeZone2.getID());
    assertNotSame(dateTimeZone2, dateTimeZone0);
    assertSame(dateTimeZone2, dateTimeZone1);
    assertFalse(dateTimeZone2.equals((Object)dateTimeZone0));

    String string2 = dateTimeZone0.getShortName((-996L), locale0);
    assertEquals("+00:00", string2);
    assertNotNull(string2);
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("UTC", dateTimeZone0.toString());
    assertEquals("UTC", dateTimeZone0.getID());
    assertEquals("deu", locale0.getISO3Language());
    assertEquals("", locale0.getCountry());
    assertEquals("de", locale0.toString());
    assertEquals("", locale0.getVariant());
    assertEquals("de", locale0.getLanguage());
    assertEquals("", locale0.getISO3Country());
    assertNotSame(dateTimeZone0, dateTimeZone2);
    assertNotSame(dateTimeZone0, dateTimeZone1);
    assertFalse(dateTimeZone0.equals((Object)dateTimeZone2));
    assertFalse(dateTimeZone0.equals((Object)dateTimeZone1));
    assertTrue(string2.equals((Object)string0));
```

```
    assertFalse(string2.equals((Object)string1));

    dateTimeZone0.hashCode();
    assertTrue(dateTimeZone0.isFixed());
    assertEquals("UTC", dateTimeZone0.toString());
    assertEquals("UTC", dateTimeZone0.getID());
    assertNotSame(dateTimeZone0, dateTimeZone2);
    assertNotSame(dateTimeZone0, dateTimeZone1);
    assertFalse(dateTimeZone0.equals((Object)dateTimeZone2));
    assertFalse(dateTimeZone0.equals((Object)dateTimeZone1));
}
```