ECOLE SUPERIEURE D'INGENIEURS DE RENNES

VALIDATION ET VÉRIFICATION
TP 3

Aymeric Pierre

Mathieu Merien

Université de Rennes

Some questions

Answer the following questions

1. Because of floating approximation, the computed value is not 1.2, but 1.20000000001. We should use Assert.assertEquals(expected, actual, delta) instead.
   ```
   assertEquals(3*0.4, 1.2,0.00001);
   ```

2. assertEqual(a,b) : check if the a and b have the same value
   assertSame(a,b) : check if a and b are located in the same memory address
   ```
   int a = 2
   assertSame(a,a);
   assertEqual(a,a);
   ```
   ```
   int a = 2;
   int b = 2;
   assertSame(a,b)
   assertSame(a,b)
   ```

3. a useful use case of the fail keyword is to test if the code fails, when it has to.
   ```
   try{
     functionThatMustFail();
     fail("No failure")
   }catch(Exception e){
     assertEqual(e, THE_ERROR_I_M_LOOKING_FOR);
   }
   ```

4. the obvious thing I see, is `assertThrows` tell me we are looking for an exception, while `@Test` don't. `@Test` don't check the type of the exception by itself, you must create some `ExpectedException`. And `assertThrows()` allow the user to check the type of exception and its details in one place.

Detecting test smells with PMD

We scanned apache/commons-collections for detached test case, and we have found some, including this one :

```
public void firstFromIterable() throws Exception {
    // Collection, entry exists
    final Bag<String> bag = new HashBag<>();
    bag.add("element", 1);
    assertEquals("element", IterableUtils.first(bag));
}
```

This is a test case, since it's located in the test folder, and use the `assertEquals()` method. However, the developer forgot the `@Test` annotation before the test.

Improvement

```
@Test
public void firstFromIterable() throws Exception {
    // Collection, entry exists
    final Bag<String> bag = new HashBag<>();
    bag.add("element", 1);
    assertEquals("element", IterableUtils.first(bag));
}
```

Balanced strings

1. the input space partition will be the following :
   - empty string
   - string with no ()[]{}
   - correct string with ()[]{}
   - correct string where ()[]{} are imbricated in each other
   - incorrect string where a closing bracket is missing
   - incorrect string where a opening bracket is missing
   - incorrect string where with this pattern ([)]
2. I wrote the test for each partition described above, I get a code coverage of 95.4%. By adding 4 more test case, I get 97.6%
3. I previously increased the code coverage by satisfying the Base Choice Coverage, the coverage remains 97.6%.
4. PIT have created 15 Mutant, and all of them has been killed. I'm very confident about the quality of my test suite.

Date I)

constructor and isValidDate

Characteristics	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
year						%4==0	%4!=0		
month	>12	<0	>12 && <0	>12 && <0	>12 && <0	2	2	[1,3, 5,7, 8,10, 12]	[4,6, 9,11]
day			<0	>31	[1...27]	28	[28,29]	[28,29, 30,31]	[28,29, 30]

I have forgot in my first iteration of the test class to add test for block 8. I found that the test was redundant for both constructor and isValidDate, so, i change the assertion of the constructor to an assertion calling isValidDate. Their is now no need to test the constructor as testing it now will only test isValidDate.

compareTo

Characteristics	**Block1**	**2**	**3**	**4**	**5**	**6**	**7**
year	>year2	<year2	=year2	=year2	=year2	=year2	=year2
month			>month2	<month2	=month2	=month2	=month2
day					>day2	<day2	=day2

nextDate

Characteristics	**Block1**	**2**	**3**
year			
month		!=12	==12

Characteristics	**Block1**	**2**	**3**
day	not end of month	end of month	end of month

previousDate

Charac-teristics	**Block1**	**2**	**3**	**4**	**5**	**6**
year						
month		month-1 in 31 days && month!=1	month-1 in 30 days	month-1 in 29 days	month-1 in 28 days	month == 1
day	not first of month	first of month	first of month	first of month	first of month	first of month

isLeapYear

Characteristics	**Block1**	**2**
year	year%4==0	year%4!=0

II)

for the test of isValidDate, I have tried to get 2 test per block, one true one false. But i also tried to limit redundant test so some block have still only one test

for every other method(except isLeapYear), the argument are a date object or two, i make the assumption that the date take into account is Valid.

for next date and previousDate i had the idea to use the isValidDate method. This allow me to simplifed the block with for exemple the fact that i can easily see the end of the month by testing isValidDate(day+1,month,year).

For every block i now only make one test, the isValidDate date is for me the crucial method to me because an error in isValidDate will surely create error in other method as the validity of y=the date is a prerequisite to use them. As such the other method don't forme need 2 test per block.

III) All my test have an oracle that respect the BCC criteria. This is because every one of my oracle are expecting a boolean expression with only one operator.

IV) On the first tests, I put assert instead of assertTrue leading to a miserable 14% coverage. After the resolution of this pb i got 95% coverage. Only stay 5 mutants. After correction only one stay, an equivalent mutant.
Mutation:

```
if(isValidDate(31,month+(normaly -)1,year)){
        return new Date(31,month-1,year);
    }
```

indeed, any month whose previous month has 31 days has the next month also with 31 days.

Mockito

The is three modules, TLSProtocol, SSLSocket and prepareSocket.

- **prepareSocket** depend on TLSProtocol, and SSLSocket

- **SSLSocket** depend on prepareSocket

Since SSLSocket is just an interface, there is nothing to test. However, prepareSocket yes. Due to cyclic dependency, I must create a stub in order to test prepareSocket.

For the two list of protocol : the list can be empty, the list can have weird protocol, and, the list can be normal. That creates 9 different case to test.

The statement coverage is : 100% for TLSSocketFactory

And the mutation score is : 92%, 13 mutant, 12 killed. The mutant who survived was very smart and remove the call to the function that trigger the test, so he can't be tested and killed

Which are great.