

TDD and JUnit

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
<http://www.berkut.mk.ua>



History

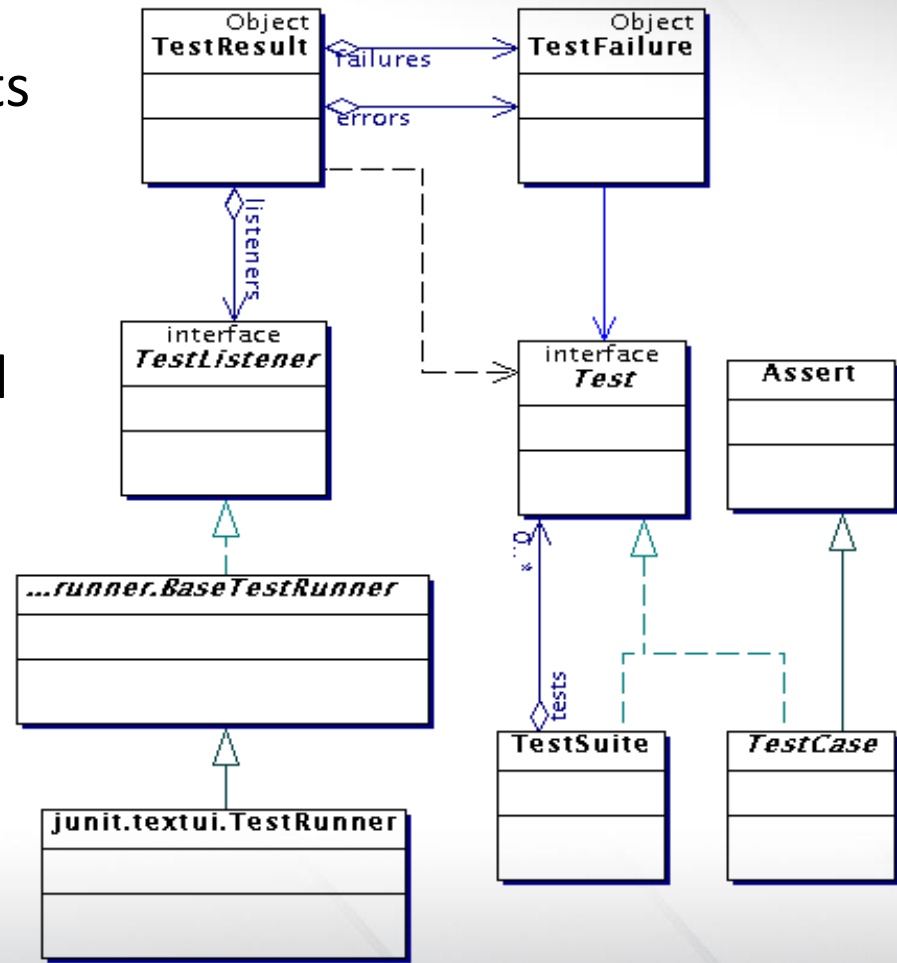
- Kent Beck developed the first xUnit automated test tool for Smalltalk in mid-90's
- Beck and Gamma (of design patterns Gang of Four) developed JUnit on a flight from Zurich to Washington, D.C.
- Martin Fowler: "Never in the field of software development was so much owed by so many to so few lines of code."
- JUnit has become the standard tool for Test-Driven Development in Java (see junit.org)
- JUnit test generators now part of many Java IDEs (IntelliJ IDEA, NetBeans, Eclipse, BlueJ, ...)

Why create a test suite?

- Obviously you have to test your code—right?
 - You can do *ad hoc* testing (running whatever tests occur to you at the moment), or
 - You can build a test suite (a thorough set of tests that can be run at any time)
- Disadvantages of a test suite
 - It's a lot of extra programming
 - True, but use of a good test framework can help quite a bit
 - You don't have time to do all that extra work
 - *False!* Experiments repeatedly show that test suites reduce debugging time more than the amount spent building the test suite
- Advantages of a test suite
 - Reduces total number of bugs in delivered code
 - Makes code much more maintainable and refactorable

Architectural overview

- JUnit test framework is a package of classes that lets you write tests for each method, then easily run those tests
- **TestRunner** runs tests and reports **TestResults**
- You test your class by extending abstract class **TestCase** (optional)
- To write test cases, you need to know and understand the **Assert** class



Writing a TestCase

- To start using JUnit, create a subclass of *TestCase*, (optional in JUnit 4 and 5) to which you add test methods
- Name of class is important – should be of the form **MyClass***Test*
- This naming convention lets TestRunner automatically find your test classes

```
import org.junit.jupiter.api.BeforeEach;
```

```
import static org.junit.jupiter.api.Assertions.*;
```

```
class MainTest {  
    @BeforeEach  
    void setUp() {  
  
    }  
}
```

Writing methods in TestCase

- Pattern follows *programming by contract* paradigm:
 - Set up **preconditions**
 - Exercise functionality being tested
 - Check **postconditions**
- Example:

```
public void testEmptyList() {  
    Bowl emptyBowl = new Bowl();  
    assertEquals("Size of an empty list should be zero.",  
        0, emptyList.size());  
    assertTrue("An empty bowl should report empty.",  
        emptyBowl.isEmpty());  
}
```
- Things to notice:
 - Specific method signature – public void **test**Whatever()
 - Coding follows pattern
 - Notice the assert-type calls...

Assert methods

- Each assert method has parameters like these:
message, expected-value, actual-value
- Assert methods dealing with floating point numbers get an additional argument, a tolerance
- Each assert method has an equivalent version that does not take a message – however, this use is not recommended because:
 - messages helps documents the tests
 - messages provide additional information when reading failure logs

Assert methods

- `assertTrue(String message, Boolean test)`
- `assertFalse(String message, Boolean test)`
- `assertNull(String message, Object object)`
- `assertNotNull(String message, Object object)`
- `assertEquals(String message, Object expected, Object actual)`
 // uses equals method
- `assertSame(String message, Object expected, Object actual)`
 // uses == operator
- `assertNotSame(String message, Object expected, Object actual)`

More stuff in test classes

- Suppose you want to test a class **Counter**
- **public class CounterTest {**
 - This is the unit test for the **Counter** class
- **public CounterTest() { } //Default constructor**
- **protected void setUp()**
 - Test *fixture* creates and initializes instance variables, etc.
- **protected void tearDown()**
 - Releases any system resources used by the test fixture
- **public void testIncrement(), public void testDecrement()**
 - These methods contain tests for the **Counter** methods **increment()**, **decrement()**, etc.
 - Note capitalization convention

JUnit tests for Counter

```
public class CounterTest {  
    Counter counter1;  
    @BeforeEach  
    protected void setUp() { // creates a test fixture  
        counter1 = new Counter();  
    }  
    @Test  
    public void testIncrement() {  
        assertTrue(counter1.increment() == 1);  
        assertTrue(counter1.increment() == 2);  
    }  
    @Test  
    public void testDecrement() {  
        assertTrue(counter1.decrement() == -1);  
    }  
}
```

Note that each test begins with a *brand new* counter

This means you don't have to worry about the order in which the tests are run

TestSuites

- TestSuites collect a selection of tests to run them as a unit
- Collections automatically use TestSuites, however to specify the order in which tests are run, write your own:

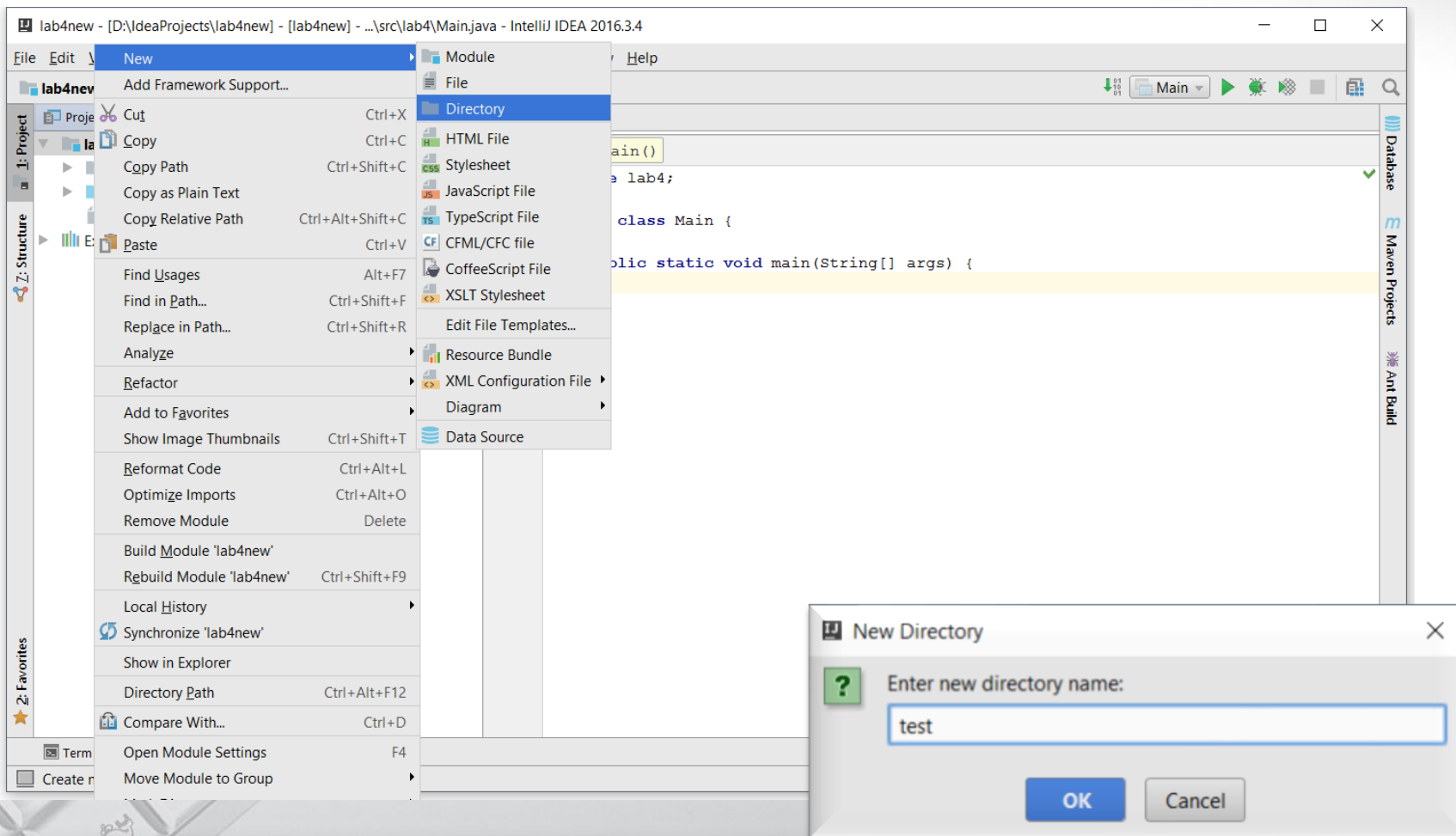
```
public static Test suite() {  
    suite.addTest(new TestBowl("testBowl"));  
    suite.addTest(new TestBowl("testAdding"));  
    return suite;  
}
```

- Should seldom have to write your own TestSuites as each method in your TestCase should be independent of all others
- Can create TestSuites that test a whole package:

```
public static Test suite() {  
    TestSuite suite = new TestSuite();  
    suite.addTestSuite(TestBowl.class);  
    suite.addTestSuite(TestFruit.class);  
    return suite;  
}
```

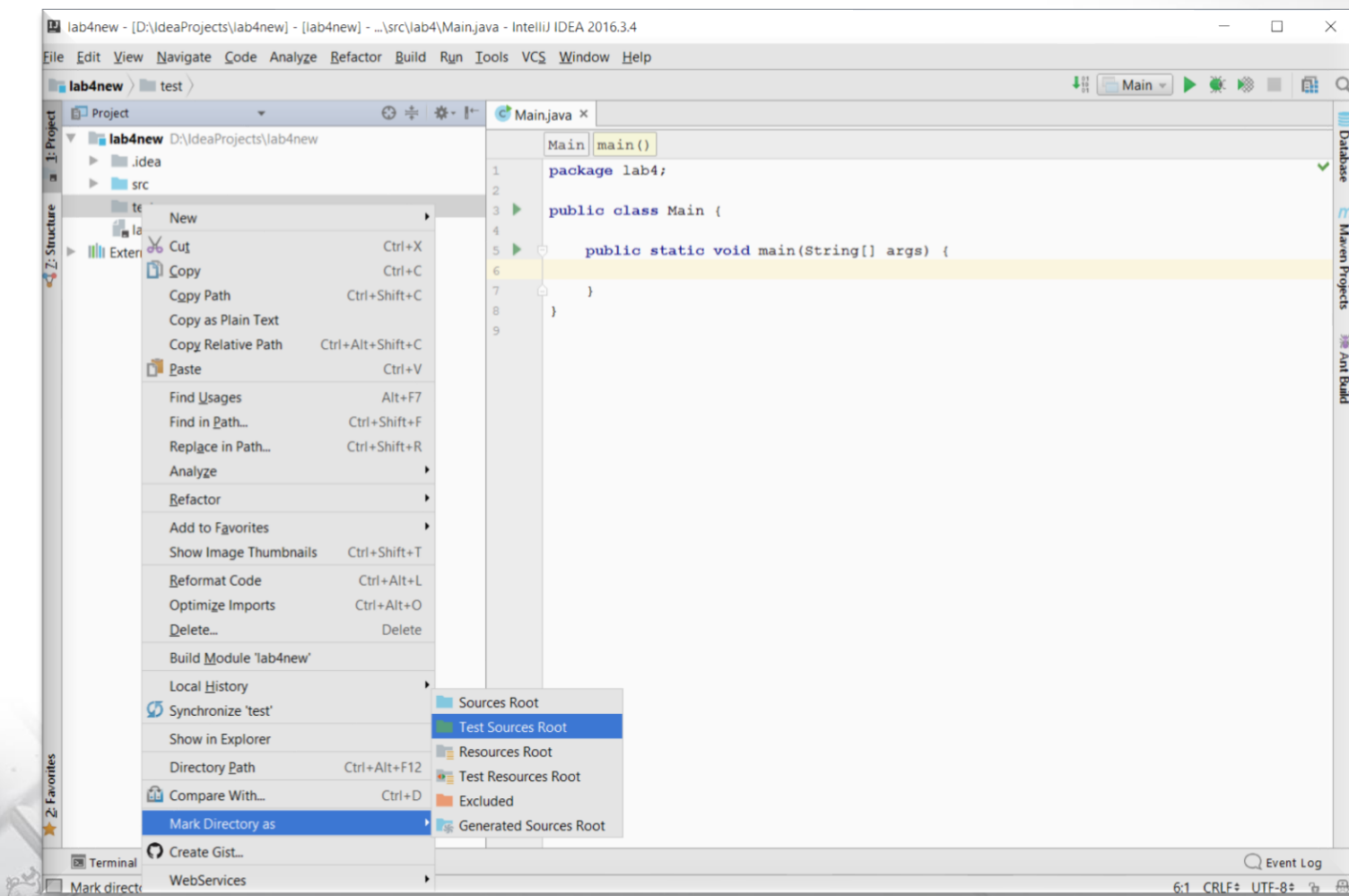
JUnit in IntelliJ IDEA

At first you have to create a directory for your tests



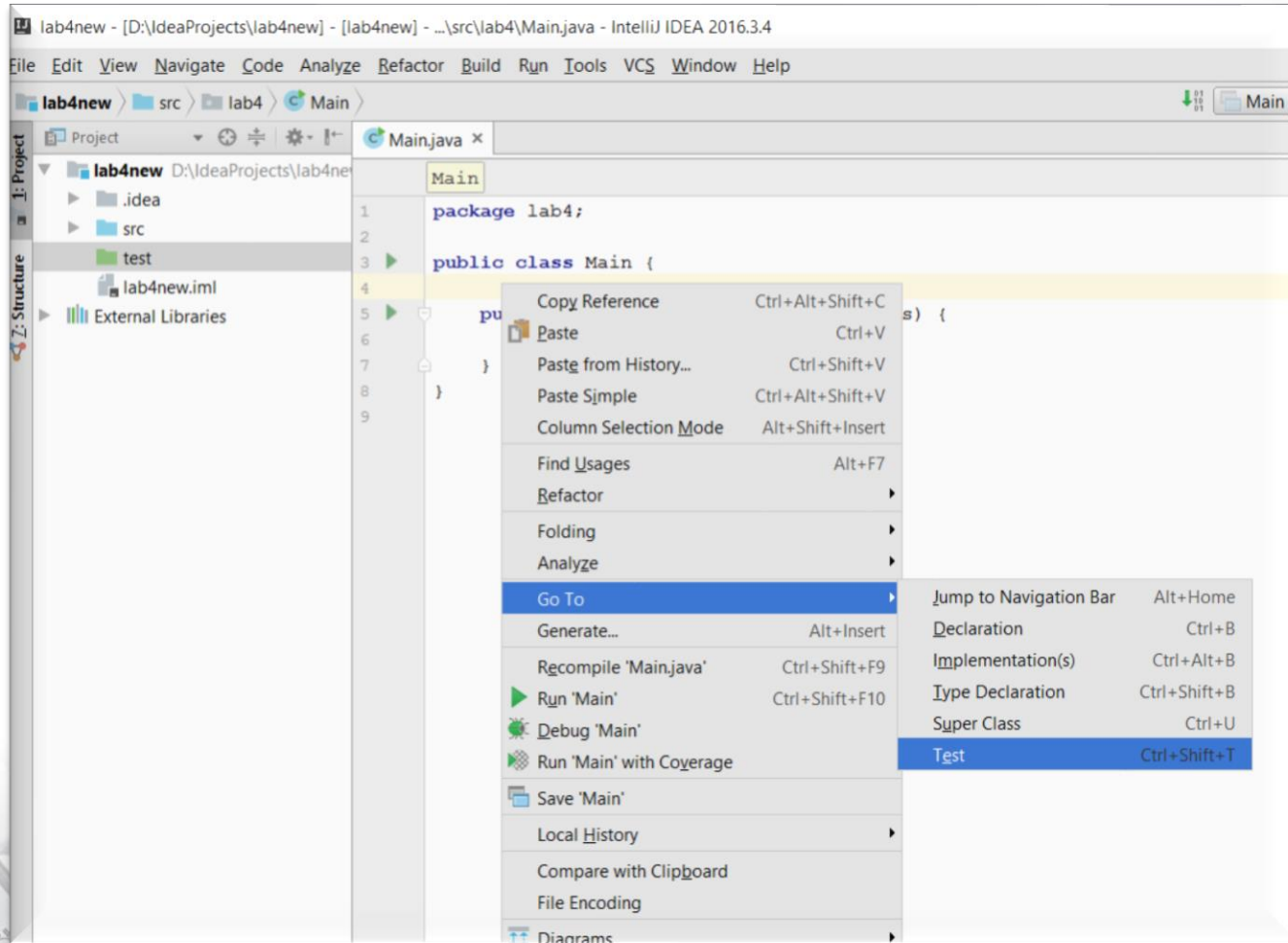
JUnit in IntelliJ IDEA

Then, mark it as Test Sources Root

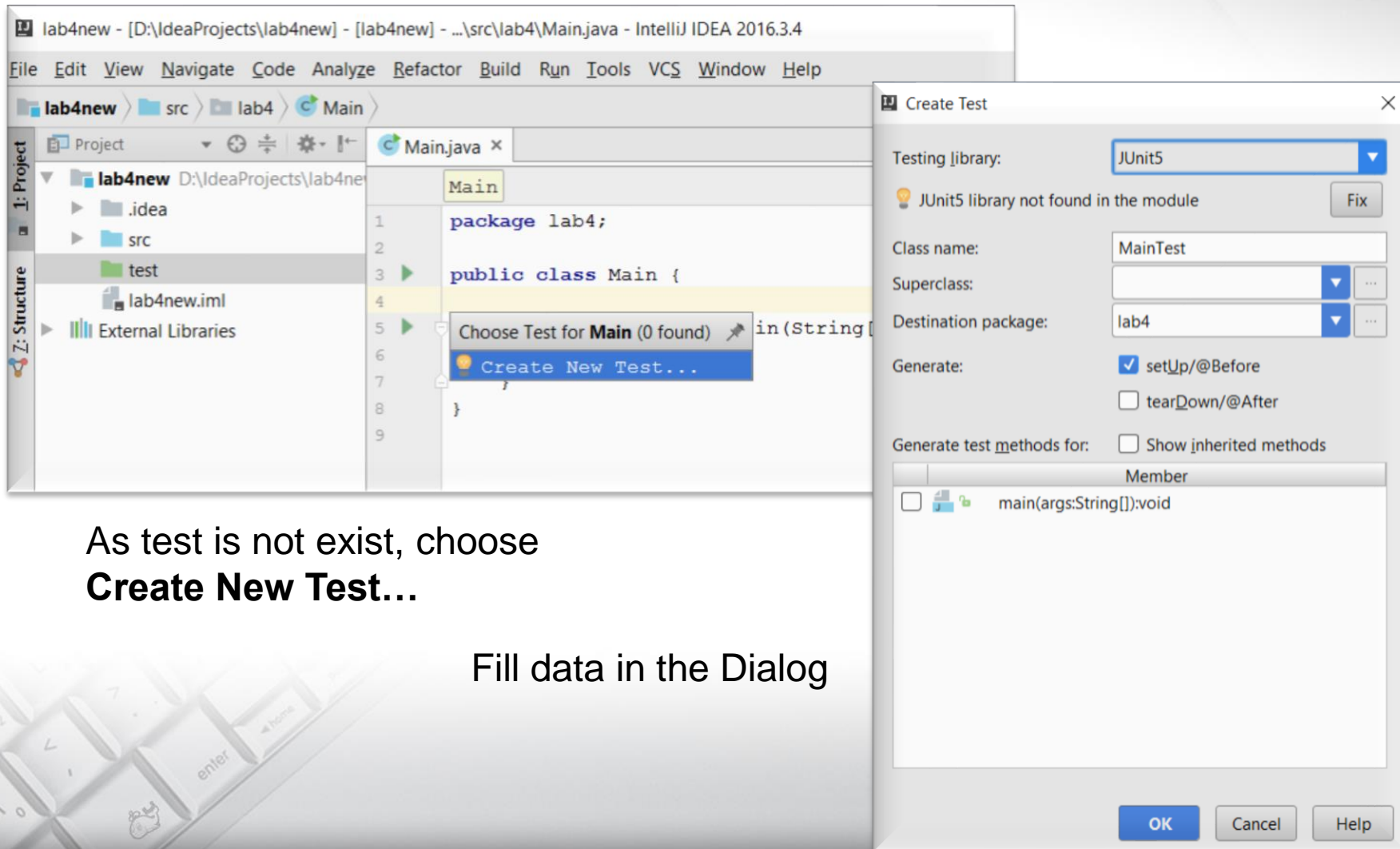


JUnit in IntelliJ IDEA

In your class choose “Go To ► Test



JUnit in IntelliJ IDEA



The screenshot shows the IntelliJ IDEA 2016.3.4 interface. The 'Main.java' file is open, showing the following code:

```
1 package lab4;  
2  
3 public class Main {  
4  
5     in (String  
6  
7  
8  
9 }
```

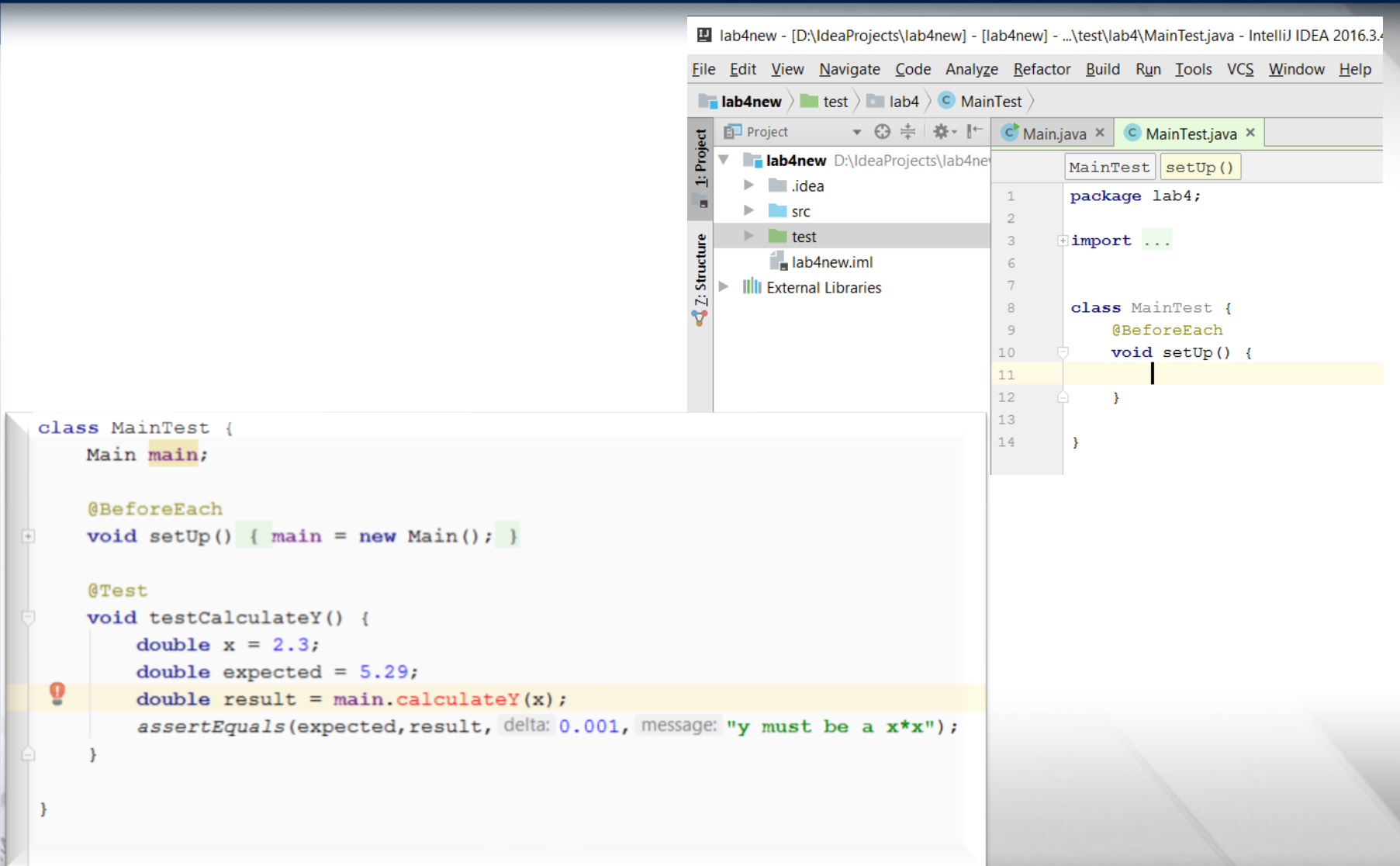
A context menu is open over the 'Main' class, with the option 'Create New Test...' selected. The 'Create Test' dialog is also open, showing the following configuration:

- Testing library: JUnit5
- JUnit5 library not found in the module (Fix button)
- Class name: MainTest
- Superclass: (empty)
- Destination package: lab4
- Generate: ☒ setUp/@Before, ☐ tearDown/@After
- Generate test methods for: ☐ Show inherited methods
- Member list: main(args:String[]):void

As test is not exist, choose **Create New Test...**

Fill data in the Dialog

JUnit in IntelliJ IDEA



lab4new - [D:\IdeaProjects\lab4new] - [lab4new] - ...test\lab4\MainTest.java - IntelliJ IDEA 2016.3.4

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

lab4new > test > lab4 > MainTest

Project D:\IdeaProjects\lab4new

- lab4new
 - .idea
 - src
 - test
 - lab4new.iml
 - External Libraries

MainTest setUp()

```
1 package lab4;
2
3 import ...
4
5
6
7
8 class MainTest {
9     @BeforeEach
10    void setUp() {
11
12    }
13
14 }
```

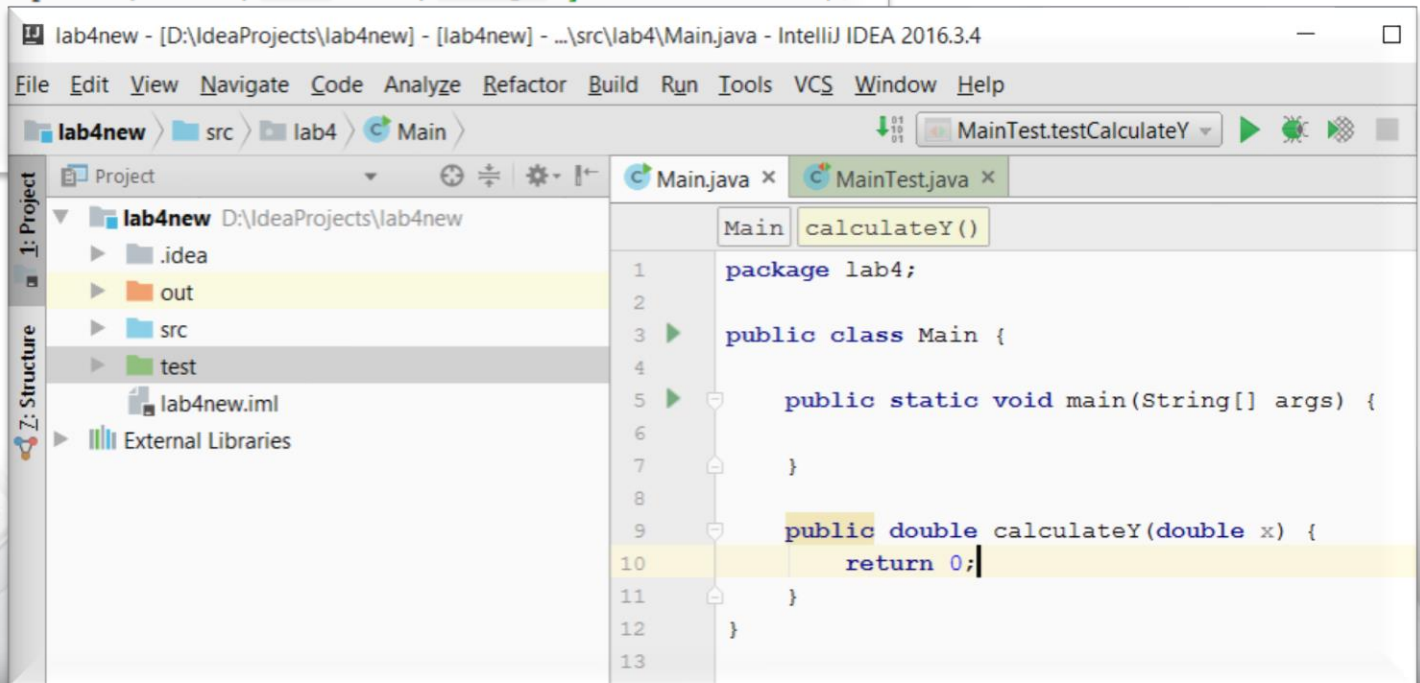
```
class MainTest {
    Main main;

    @BeforeEach
    void setUp() { main = new Main(); }

    @Test
    void testCalculateY() {
        double x = 2.3;
        double expected = 5.29;
        double result = main.calculateY(x);
        assertEquals(expected, result, delta: 0.001, message: "y must be a x*x");
    }
}
```

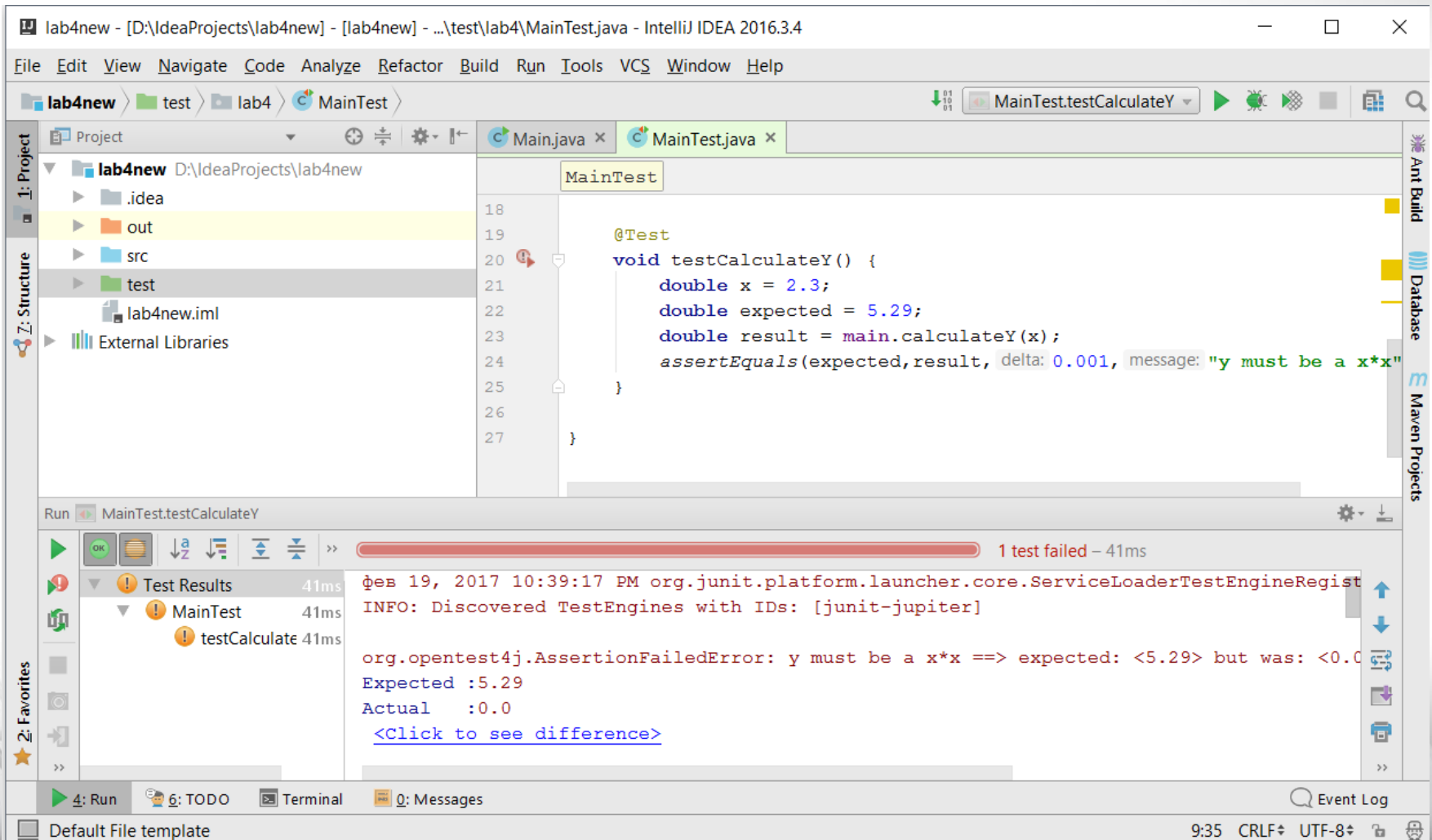

JUnit in IntelliJ IDEA

```
class MainTest {  
    Main main;  
  
    @BeforeEach  
    void setUp() { main = new Main(); }  
  
    @Test  
    void testCalculateY() {  
        double x = 2.3;  
        double expected = 5.29;  
        double result = main.calculateY(x);  
        assertEquals(expected, result, delta: 0.001, message: "y must be a x*x");  
    }  
}
```



JUnit in IntelliJ IDEA

Run test of the generated method. It fails



The screenshot shows the IntelliJ IDEA interface with a project named 'lab4new'. The 'Project' view on the left shows the directory structure: 'lab4new' (D:\IdeaProjects\lab4new) containing '.idea', 'out', 'src', 'test', 'lab4new.iml', and 'External Libraries'. The 'MainTest.java' file is open in the editor, showing a JUnit test method:

```
18  
19  
20 @Test  
21 void testCalculateY() {  
22     double x = 2.3;  
23     double expected = 5.29;  
24     double result = main.calculateY(x);  
25     assertEquals(expected, result, delta: 0.001, message: "y must be a x*x")  
26 }  
27
```

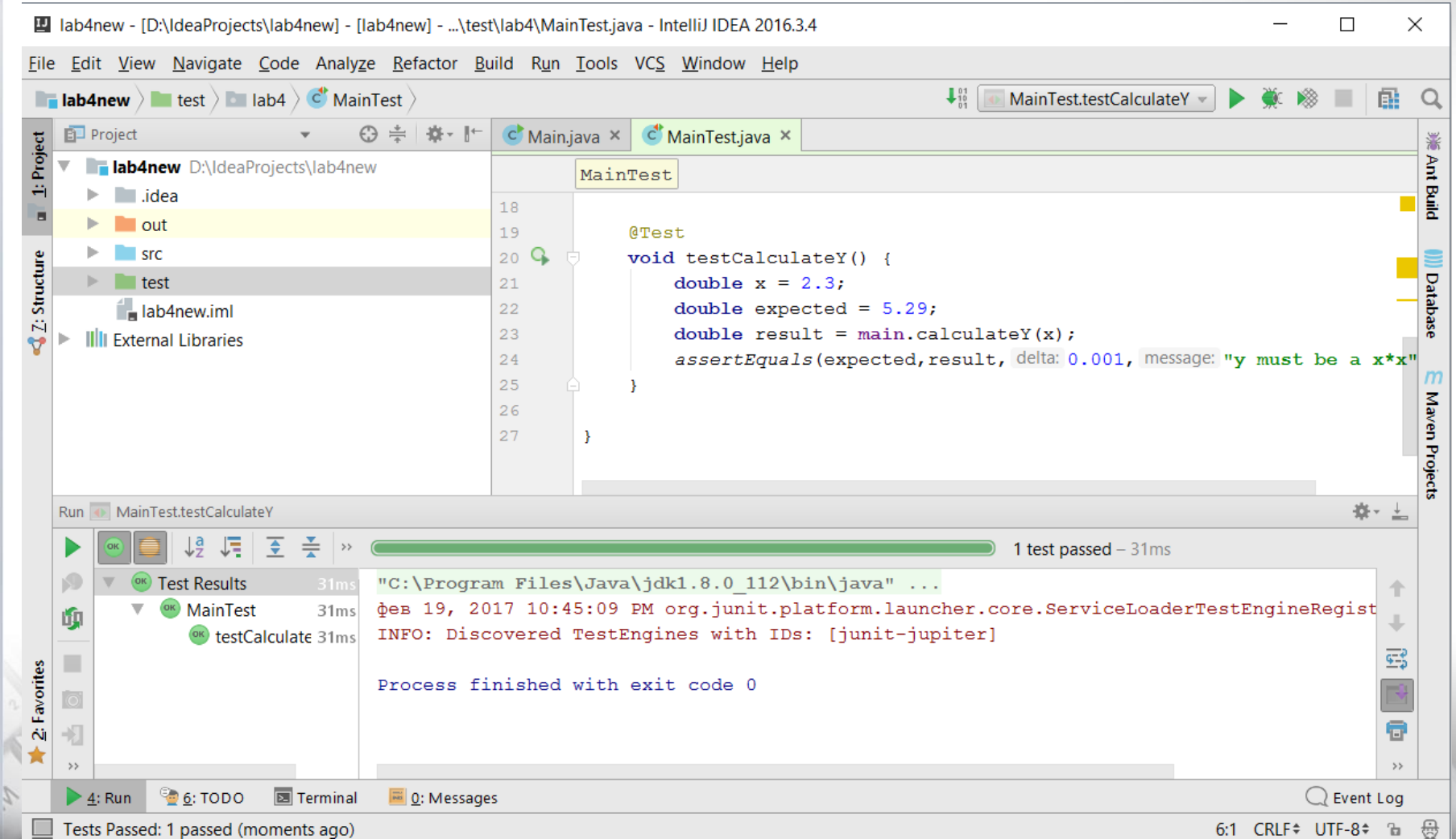
The 'Run' button is highlighted, and the 'Run' toolbar shows the test results: '1 test failed - 41ms'. The 'Test Results' panel shows the test 'testCalculate' failed with a duration of 41ms. The 'Run' panel shows the following output:

```
Feb 19, 2017 10:39:17 PM org.junit.platform.launcher.core.ServiceLoaderTestEngineRegistry  
INFO: Discovered TestEngines with IDs: [junit-jupiter]  
  
org.opentest4j.AssertionFailedError: y must be a x*x ==> expected: <5.29> but was: <0.0>  
Expected :5.29  
Actual   :0.0  
<Click to see difference>
```

The status bar at the bottom shows '4: Run', '6: TODO', 'Terminal', '0: Messages', and 'Event Log'. The system clock shows '9:35 CRLF UTF-8'.

JUnit in IntelliJ IDEA

Write correct method body. Run test of the generated method. It should be OK



The screenshot shows the IntelliJ IDEA 2016.3.4 interface. The top toolbar has a green play button icon. The main editor displays the `MainTest.java` file with the following code:

```
18  
19  
20 @Test  
21 void testCalculateY() {  
22     double x = 2.3;  
23     double expected = 5.29;  
24     double result = main.calculateY(x);  
25     assertEquals(expected, result, delta: 0.001, message: "y must be a x*x")  
26 }  
27
```

The left sidebar shows the project structure for `lab4new`, with the `test` directory selected. The bottom panel shows the Run configuration for `MainTest.testCalculateY`. The Run toolbar shows a green play button icon. The Run output shows the test results:

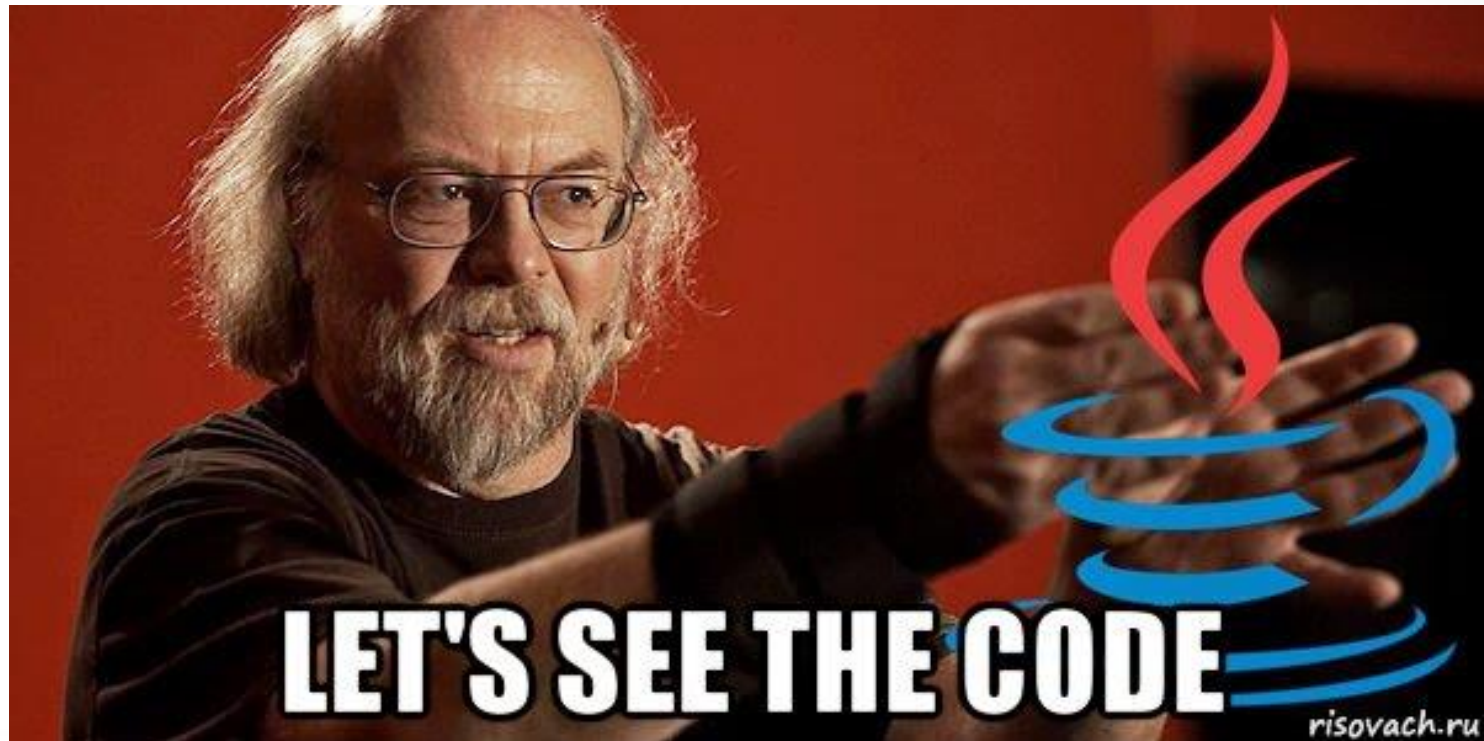
```
Run MainTest.testCalculateY  
1 test passed - 31ms  
Test Results  
MainTest 31ms  
testCalculate 31ms  
Process finished with exit code 0
```

The status bar at the bottom indicates "Tests Passed: 1 passed (moments ago)".

More Information

- <http://www.junit.org>
 - Download of JUnit
 - Lots of information on using JUnit
- <http://sourceforge.net/projects/cppunit>
 - C++ port of Junit
- <http://www.thecoadletter.com>
 - Information on Test-Driven Development

Example





НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ
КОРАБЛЕБУДУВАННЯ
ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?



TDD and JUnit

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
<http://www.berkut.mk.ua>

