

# OODP– Session 4 - Tutorial

## Session times

PT group 1 – Monday	18:00-21:00	room: Malet 403	
PT group 2 – Thursday	18:00-21:00	room: Malet 407	
FT	- Tuesday	13:30-17:00	room: Malet 404

Email: [oded@dcs.bbk.ac.uk](mailto:oded@dcs.bbk.ac.uk)

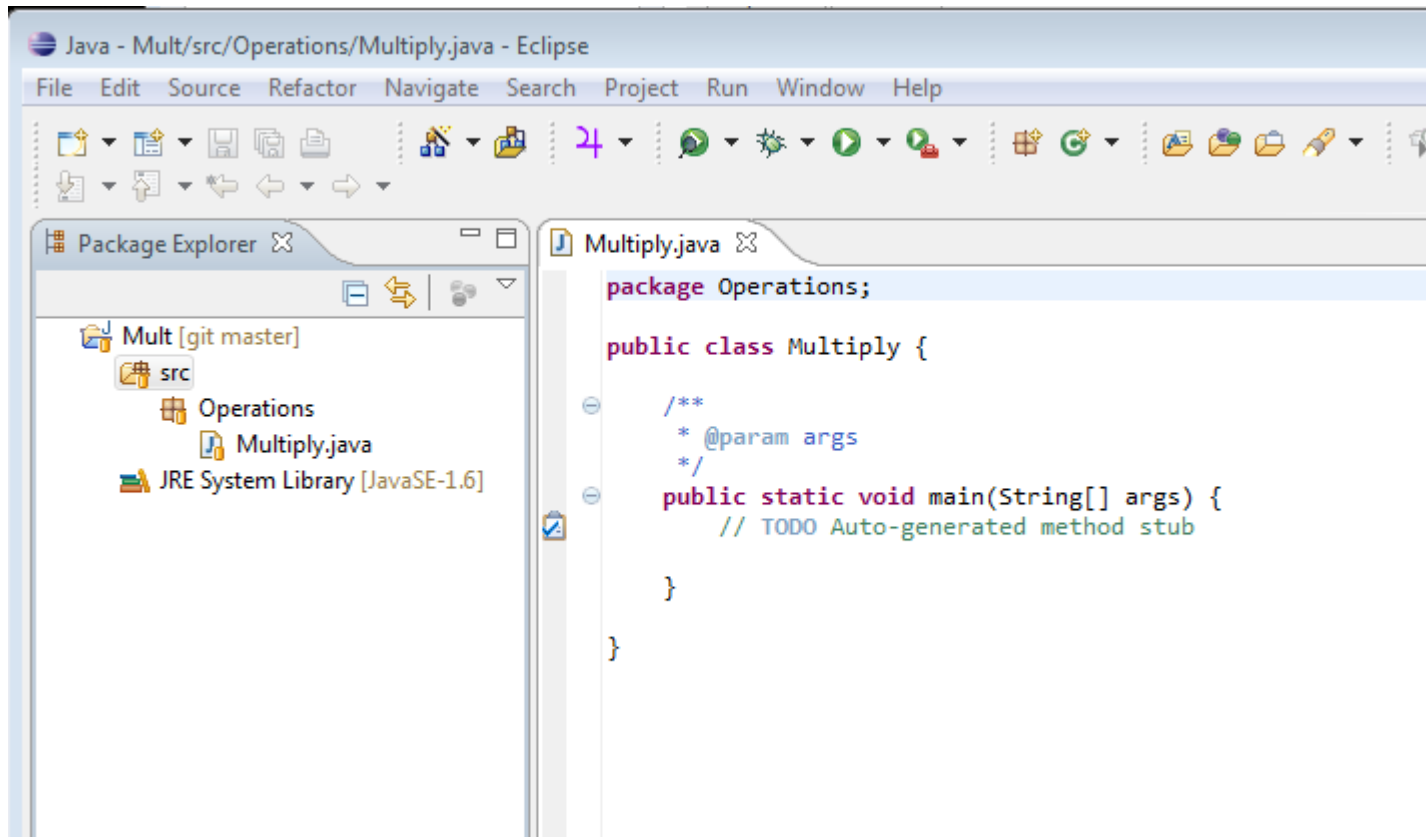
Web Page: <http://www.dcs.bbk.ac.uk/~oded>

Visiting Hours: [Tuesday 17:00 to 19:00](#)

# Our First Test

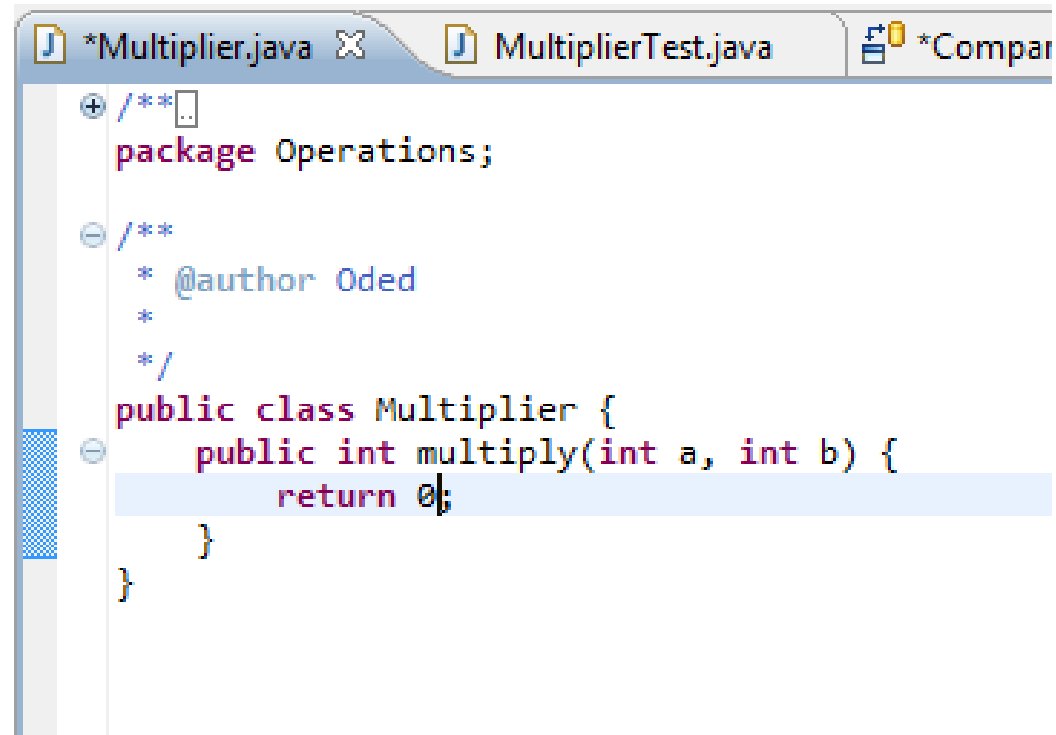
# Write your own main

Don't bother adding any coded



# Add a class dedicated to a binary operation

I chose to multiply (didn't bother with the code)



```

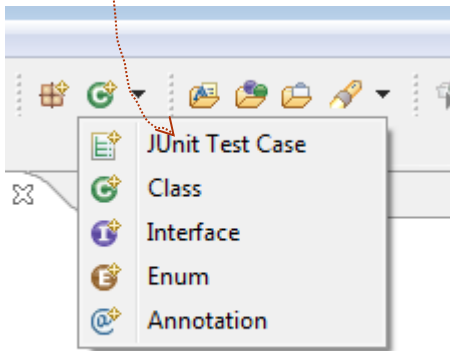
+ /**
package Operations;

- /**
 * @author Oded
 *
 */
public class Multiplier {
-     public int multiply(int a, int b) {
         return 0;
     }
}

```

# Lets start testing

Pick this



New JUnit Test Case

**JUnit Test Case**

⚠ This package name is discouraged. By convention, package names usually start with a lowercase letter

New JUnit 3 test  New JUnit 4 test

Source folder: Mult/src

Package: Operations

Name: MultiplierTest

Superclass: java.lang.Object

Which method stubs would you like to create?

setUpBeforeClass()  tearDownAfterClass()  
 setUp()  tearDown()  
 constructor

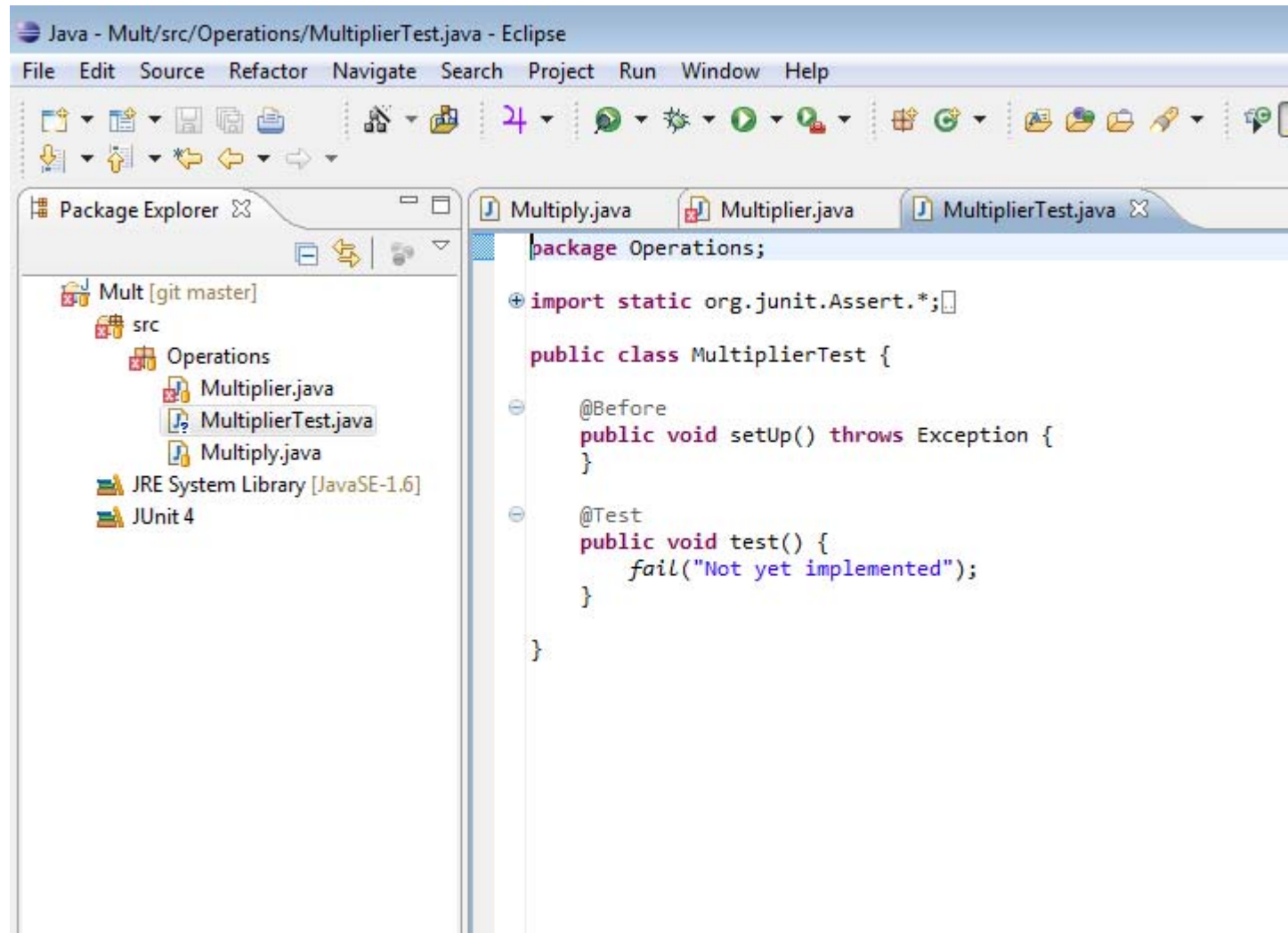
Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

Class under test: Operations.Multiplier

# Our new test

Known also



The screenshot shows the Eclipse IDE interface. The title bar reads "Java - Mult/src/Operations/MultiplierTest.java - Eclipse". The menu bar includes "File", "Edit", "Source", "Refactor", "Navigate", "Search", "Project", "Run", "Window", and "Help". The toolbar contains various icons for file operations and development tools. The Package Explorer on the left shows a project named "Mult [git master]" with a "src" folder containing an "Operations" package. Inside "Operations", there are three files: "Multiplier.java", "MultiplierTest.java" (which is selected), and another "Multiply.java". Below the project are "JRE System Library [JavaSE-1.6]" and "JUnit 4". The main editor window shows the code for "MultiplierTest.java":

```
package Operations;

import static org.junit.Assert.*;

public class MultiplierTest {

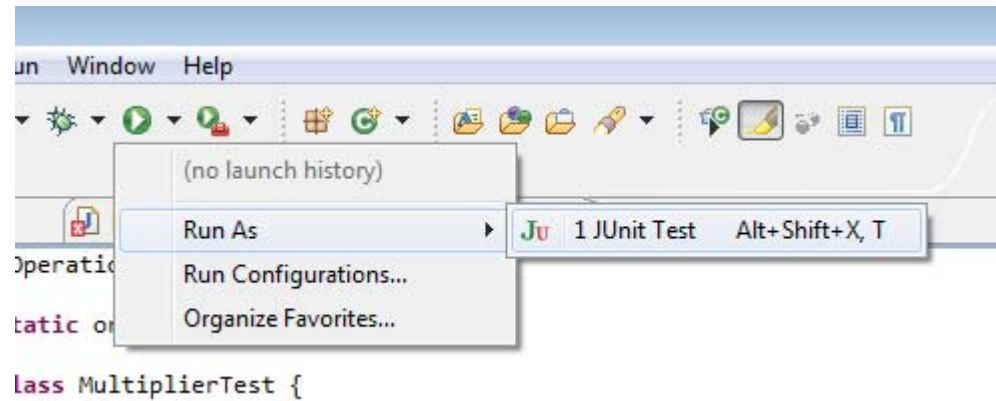
    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

# Running Our First Test

Known also



# Test Failed



Java - Mult/src/Operations/MultiplierTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 0.032 seconds

Runs: 1/1 Errors: 0 Failures: 1

Operations.MultiplierTest [Runner: JUnit 4] (0.006 s)

test (0.006 s)

Failure Trace

java.lang.AssertionError: Not yet implemented  
at Operations.MultiplierTest.test(MultiplierTest.java:16)

```
package Operations;
import static org.junit.Assert.*;

public class MultiplierTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void test() {
        fail("Not yet implemented");
    }
}
```

@ Javadoc Declaration

Operations.Multiplier

Author:  
Oded



# **Stuff we Saw on the Way**

**(to our first test)**

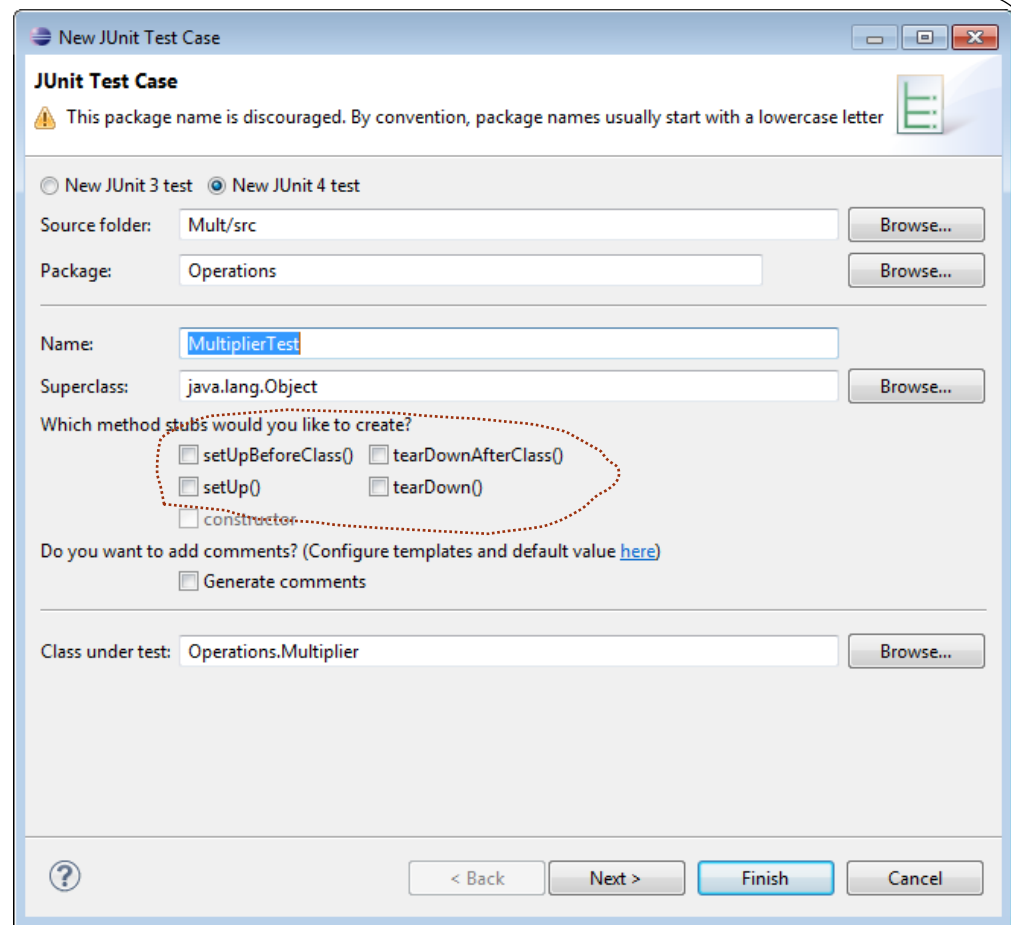
# Method stubs

**SetUpBeforeClass ()** – adds a method that is performed before all tests (we can run more than one test at a time) we might want to set up a database etc.

**SetUp()** – adds a method that is performed before the test, used to set up the test environment

**TearDownAfterClass()** – adds a method that is performed after all tests

**TearDown()** – adds a method that is performed after the test



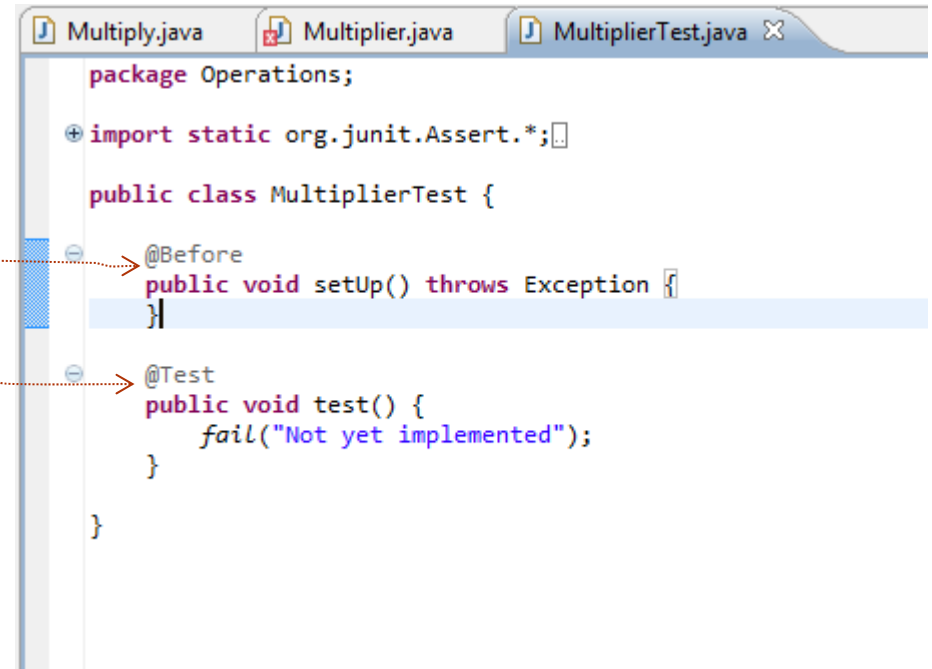
# Method Stubs Manifestation - Annotations

Annotations – are data about the program that is not part of the program itself

Uses of annotations – instruction for compiler, runtime processing, Junit 4

Before method

Test method



```
package Operations;

import static org.junit.Assert.*;

public class MultiplierTest {

    @Before
    public void setUp() throws Exception {

    }

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

# But Our Test is Empty - Assertions

`@Test` - this is a test method

`@Before` - this is a before method

`@After` - this is teardown method

`@BeforeClass` - this a before class method

`@AfterClass` - this is teardown after class

`@Ignore` – ignore this method, when you don't want one of the above executed

`@Test(expected=IllegalArgumentException.class)` - does the method throw the named exception (Exception?)

`@Test(timeout=10)` fail after 10 millisecond – so the test does not run forever

(add timeout to your code)

# Assertions

- `fail([String])`- test failed
- `AssertTrue(true)`- test passed
- `assertEquals([String], expected, actual)`- Check if values of expected and actual are the same.
- `assertNull([String], object)` – Check reference null
- `assertNotNull([String], object)` – Check if reference not null
- `assertSame([String], expected, actual)` – Check if reference to the same object
- `assertNotSame([String], expected, actual)` - Check if reference to different objects
- `assertTrue([String], boolean condition)` – Check if Boolean condition is true

# Our First TDD Iteration

# Real Test

- The new code is on the left (result of comparing to HEAD)

```
Java Source Compare
```

Local: MultiplierTest.java

```
import org.junit.Test;

public class MultiplierTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test(timeout = 10)
    public void test() {
        Multiplier mult = new Multiplier();
        assertEquals("Can't multiply", 15, mult.multipl
    }

}
```

MultiplierTest.java 2ae6078... (oded)

```
import org.junit.Before;
import org.junit.Test;

public class MultiplierTest {

    @Before
    public void setUp() throws Exception {
    }

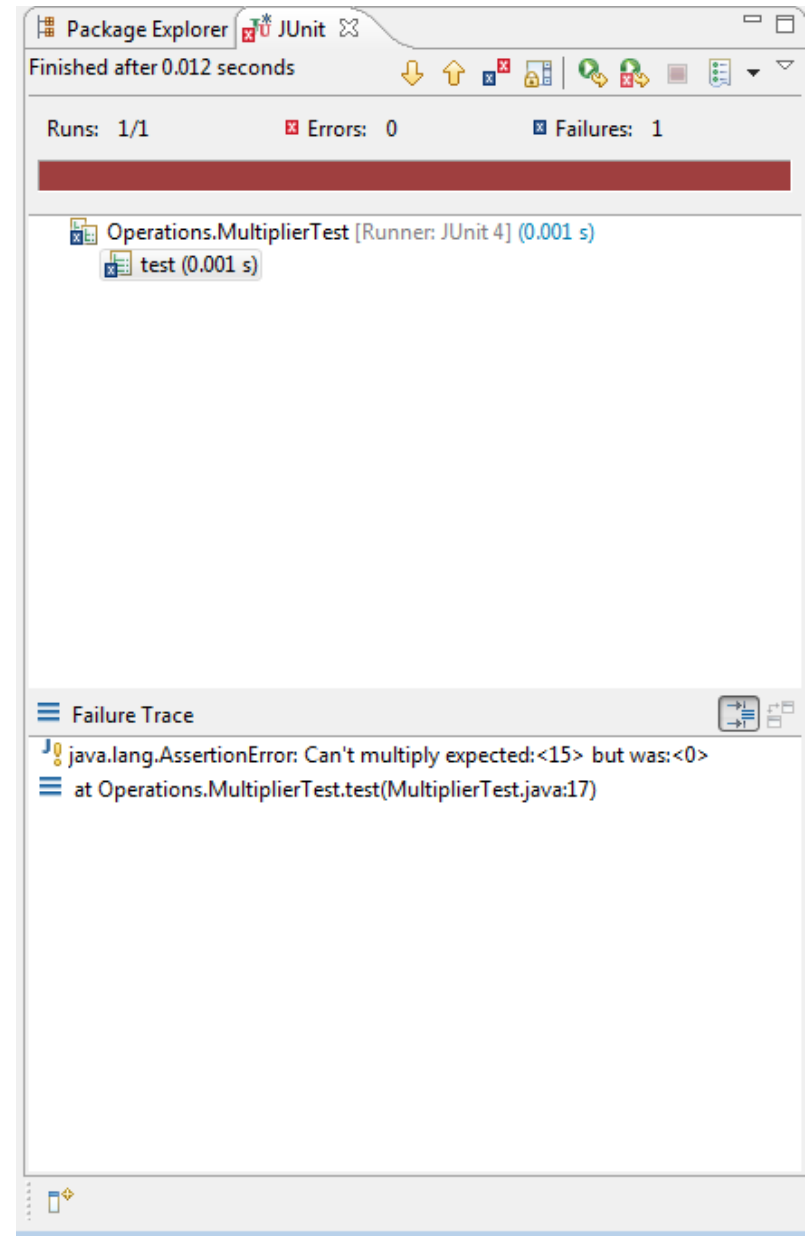
    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

# Real Test result

- It failed

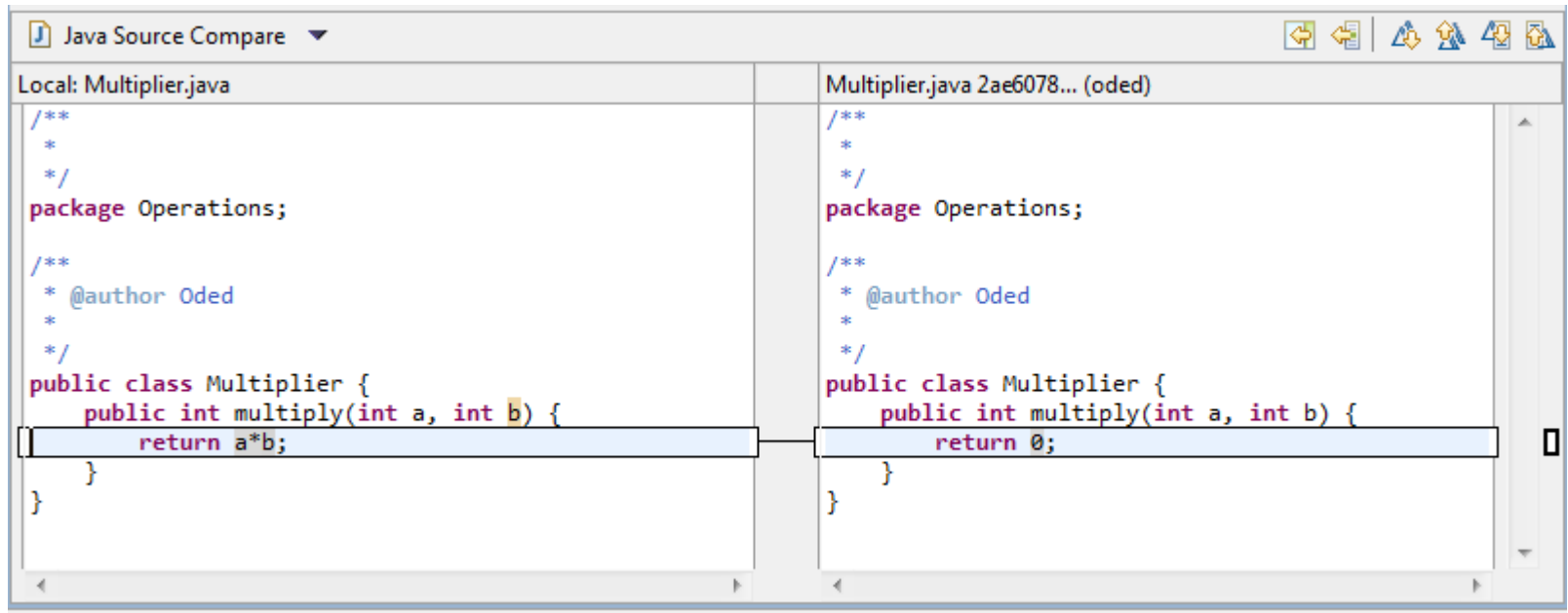
TDD says it should – cause we haven't written the implementation!





# The implementation

- Implementation on the left

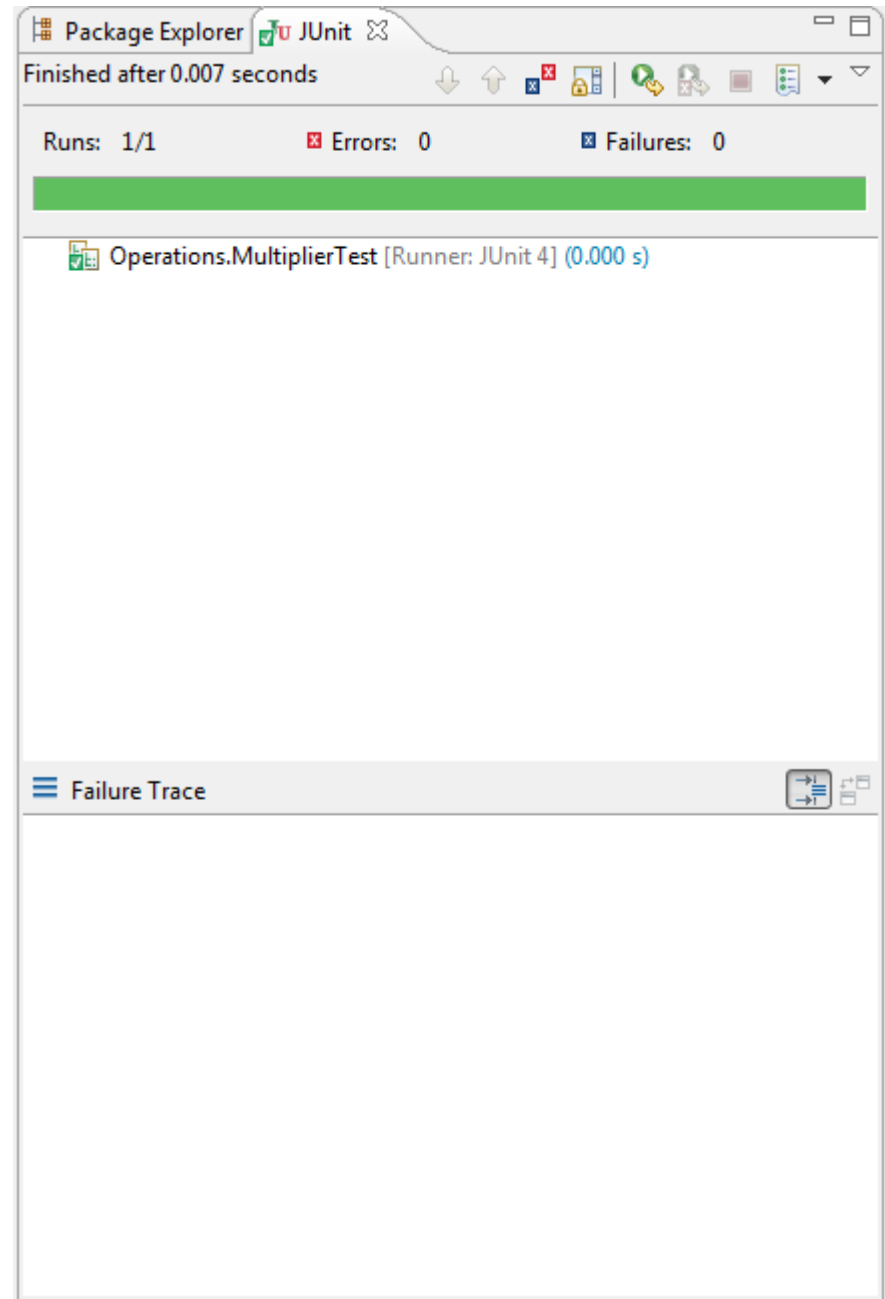


```
Java Source Compare
```

Local: Multiplier.java	Multiplier.java 2ae6078... (oded)
<pre>/**  *  */ package Operations;  /**  * @author Oded  *  */ public class Multiplier {     public int multiply(int a, int b) {         return a*b;     } }</pre>	<pre>/**  *  */ package Operations;  /**  * @author Oded  *  */ public class Multiplier {     public int multiply(int a, int b) {         return 0;     } }</pre>

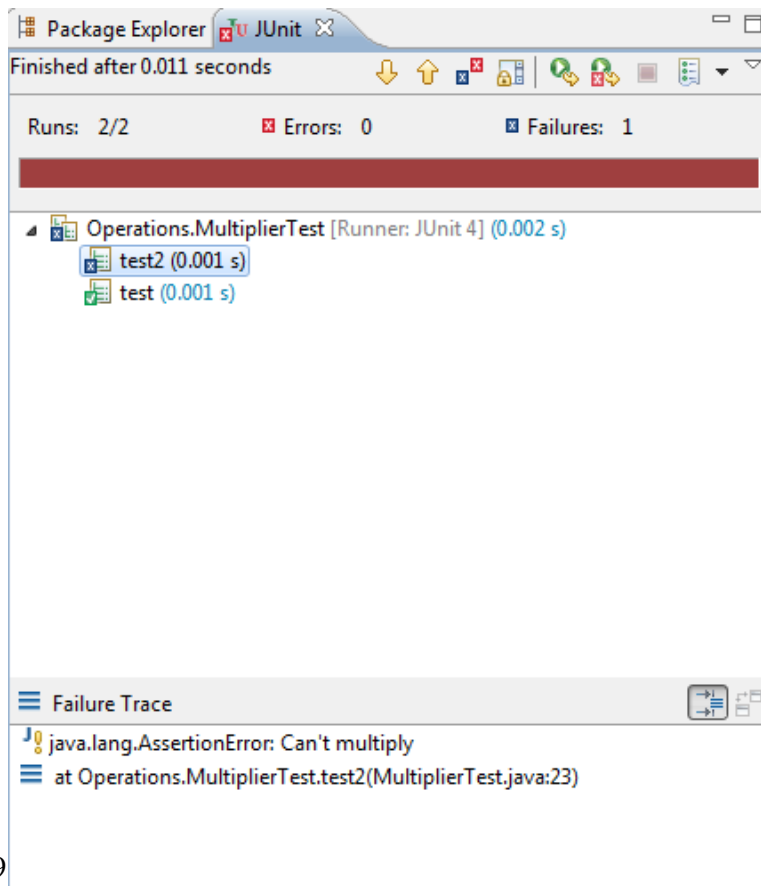
# Running our test again

- Now it passed  
(surprise)



# Another Test

- One passed One Failed



The screenshot shows the JUnit test runner interface. At the top, it says "Finished after 0.011 seconds". Below that, it displays "Runs: 2/2", "Errors: 0", and "Failures: 1". A red progress bar indicates the test results. The test results list shows "Operations.MultiplierTest [Runner: JUnit 4] (0.002 s)" with two sub-tests: "test2 (0.001 s)" which failed (indicated by a red 'x' icon) and "test (0.001 s)" which passed (indicated by a green checkmark icon). At the bottom, the "Failure Trace" section shows the error: "java.lang.AssertionError: Can't multiply" at "Operations.MultiplierTest.test2(MultiplierTest.java:23)".

```
Multiplier.java Multiplier.java MultiplierTest.java X
package Operations;

import static org.junit.Assert.*;

public class MultiplierTest {

    @Before
    public void setUp() throws Exception {
    }

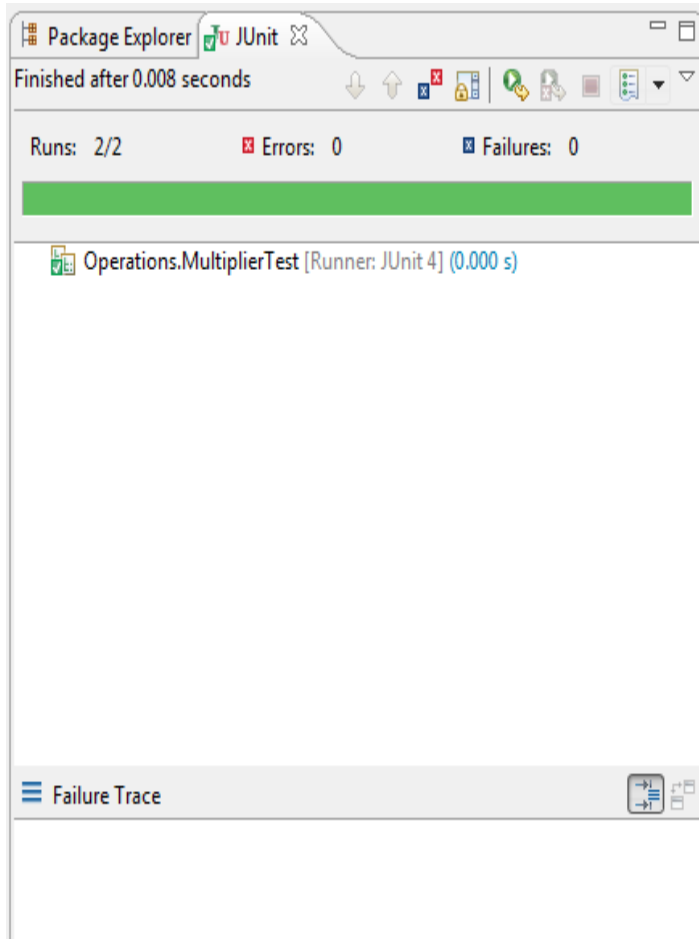
    @Test(timeout = 10)
    public void test() {
        Multiplier mult = new Multiplier();
        assertEquals("Can't multiply",15,mult.multiply(3,5));
    }

    @Test(timeout = 10)
    public void test2() {
        Multiplier mult = new Multiplier();
        assertTrue("Can't multiply", (mult.multiply(3,6) != 18));
    }

}
```

# After Fixing

- One passed One Failed



```
package Operations;

import static org.junit.Assert.*;

public class MultiplierTest {

    @Before
    public void setUp() throws Exception {
    }

    @Test(timeout = 10)
    public void test() {
        Multiplier mult = new Multiplier();
        assertEquals("Can't multiply", 15, mult.multiply(3, 5));
    }

    @Test(timeout = 10)
    public void test2() {
        Multiplier mult = new Multiplier();
        assertTrue("Can't multiply", (mult.multiply(3, 6) != 15));
    }
}
```

# Note

Here we wrote one of the tests after we implemented the code.

According to TDD we should have tried to avoid this.

# **Our Second TDD Iteration Refactoring**

# Extract Interface

1. Rename “multiply” to “oper”
2. Extract interface

```
Multiply.java Multiplier.java MultiplierTest.java
+ /**
package Operations;

- /**
 * @author Oded
 *
 */
public class Multiplier implements Operation {
    /* (non-Javadoc)
     * @see Operations.Operation#oper(int, int)
     */
    @Override
    public int oper(int a, int b) {
        return a*b;
    }
}
```

```
Multiply.java Multiplier.java MultiplierTest.java
package Operations;

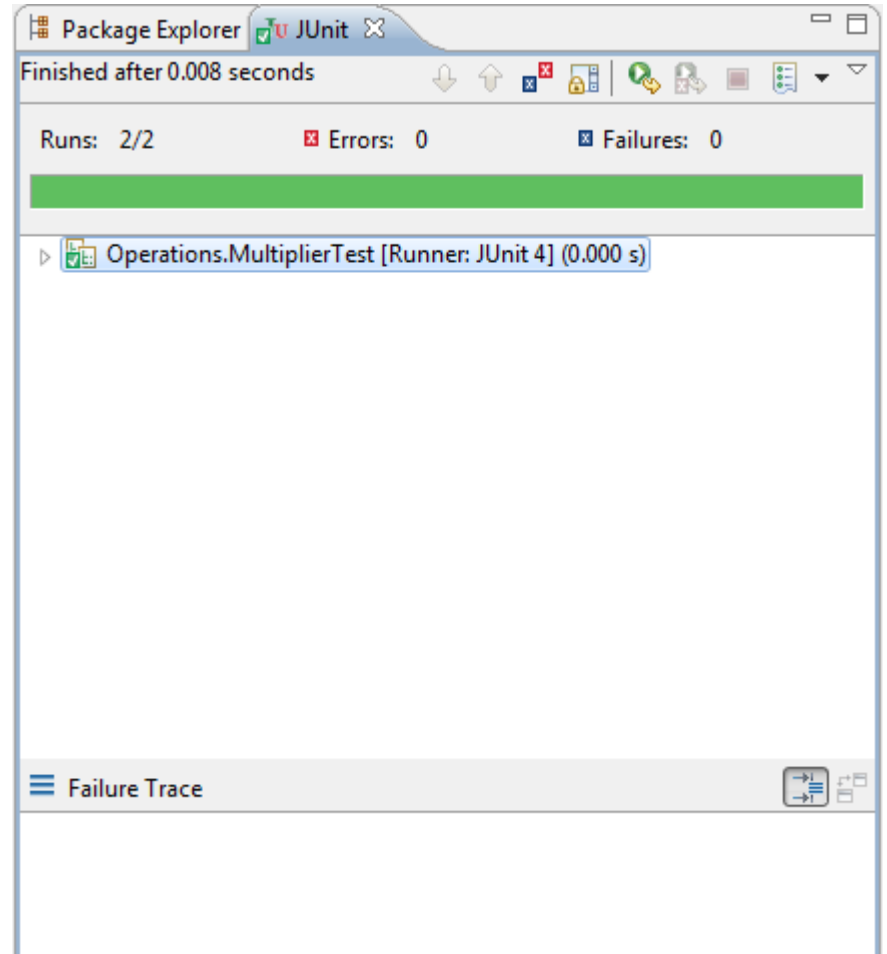
public interface Operation {

    public abstract int oper(int a, int b);

}
```

# Run Test

Surprise it passed






# **Our Third TDD Iteration**

## **Adding a Class**

# New Object Adder

New Java Class

**Java Class**

 This package name is discouraged. By convention, package names usually start with a lowercase letter

Source folder:

Package:

Enclosing type:

---

Name:

Modifiers:  public  default  private  protected  
 abstract  final  static

Superclass:

Interfaces:

Which method stubs would you like to create?

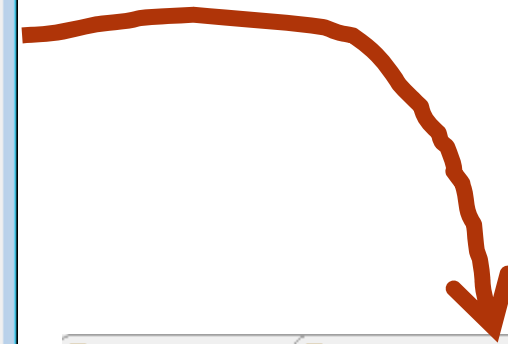
public static void main(String[] args)

Constructors from superclass

Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments



```
Multiply.java  Multiplier.java  MultiplierTest.java
package Operations;

public class Adder implements Operation {

    @Override
    public int oper(int a, int b) {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

# New Test,

```
Multiply.java  Multiplier.java  MultiplierTest.java
+ /**
package Operations;

+ import static org.junit.Assert.*;

- /**
 * @author Oded
 *
 */
public class AdderTest {

-     /**
 * @throws java.lang.Exception
 */
-     @Before
public void setUp() throws Exception {
    }

-     @Test
public void test() {
    fail("Not yet implemented");
}
}
```

Finished after 0.012 seconds

Runs: 3/3      ✖ Errors: 0      ✖ Failures: 1

Operations.AdderTest [Runner: JUnit 4] (0.000 s)  
    ✖ test (0.000 s)

Operations.MultiplierTest [Runner: JUnit 4] (0.002 s)

Failure Trace

java.lang.AssertionError: Not yet implemented  
at Operations.AdderTest.test(AdderTest.java:26)

# After fixing test and then writing code

```
Multiplied.java  Multiplier.java  MultiplierTest.java
+ /**
package Operations;

+ import static org.junit.Assert.*;

- /**
 * @author Oded
 *
 */
public class AdderTest {

- /**
 * @throws java.lang.Exception
 */
- @Before
public void setUp() throws Exception {
}

- @Test
public void test() {
    Adder a = new Adder();
    assertEquals("can't add",8,a.oper(3,5));
}
}
```

```
Multiplied.java  Multiplier.java  MultiplierTest.java
package Operations;

public class Adder implements Operation {

    @Override
    public int oper(int a, int b) {
        // TODO Auto-generated method stub
        return a+b;
    }
}
```

```
Finished after 0.009 seconds
Runs: 3/3      Errors: 0      Failures: 0

Operations.AdderTest [Runner: JUnit 4] (0.000 s)
Operations.MultiplierTest [Runner: JUnit 4] (0.000 s)
```

# Lets make the test a bit more interesting

The screenshot shows an IDE with a Java test class named `AdderTest`. The code includes a package declaration, imports, a class definition with a `setUp()` method and a `test()` method. The `test()` method contains an `assertEquals` call. A context menu is open over the `assertEquals` line, with `Surround With` selected. A submenu is visible, showing `Try/catch Block` and `1 do (do while statement)`. In the bottom right, a test runner window shows the test `test` passing in 0.000 seconds.

```
/**  
package Operations;  
  
import static org.junit.Assert.*;  
  
/**  
 * @author Oded  
 */  
public class AdderTest {  
  
    /**  
     * @throws java.lang.Exception  
     */  
    @Before  
    public void setUp() throws Exception {  
    }  
  
    @Test  
    public void test() {  
        Adder a = new Adder();  
        assertEquals("can't add", 8, a.oper(8, 8));  
    }  
}
```

Quick Outline Ctrl+O  
Quick Type Hierarchy Ctrl+T  
Open With  
Show In Alt+Shift+W  
Cut Ctrl+X  
Copy Ctrl+C  
Copy Qualified Name  
Paste Ctrl+V  
Quick Fix Ctrl+1  
Source Alt+Shift+S  
Refactor Alt+Shift+T  
Surround With Alt+Shift+Z  
Local History  
References  
Declarations  
Add to Snippets...  
Find Bugs  
Run As  
Debug As

Try/catch Block  
1 do (do while statement)  
2 for (for each statement)  
3 for (for statement)  
4 for (for loop statement)  
5 for (for loop statement with else)  
6 for (for loop statement with else and finally)  
7 for (for loop statement with else and finally and catch)  
8 for (for loop statement with else and finally and catch and finally)  
9 for (for loop statement with else and finally and catch and finally and catch)  
10 for (for loop statement with else and finally and catch and finally and catch and finally)

Finished after 0.01 seconds  
Runs: 1/1 Errors: 0 Failures:  
Operations.AdderTest [Runner: JUnit 4] (0.000 s)  
test (0.000 s)

# Lets make the test a bit more interesting



```
Multiplier.java | MultiplierTest.java | Adder.java | *AdderTest.java | AllTests.ja
+ /**
  package Operations;

+ import static org.junit.Assert.*;

- /**
  * @author Oded
  *
  */
  public class AdderTest {

-   /**
    * @throws java.lang.Exception
    */
-   @Before
    public void setUp() throws Exception {
    }

-   @Test
    public void test() {
        Adder a = new Adder();
        while (condition) {
            assertEquals("can't add", 8, a.oper(3, 5));
        }
    }
  }
```

# Lets make the test a bit more interesting

```
Multiplier.java MultiplierTest.java Adder.java x
```

```
package Operations;

public class Adder implements Operation {

    @Override
    public int oper(int a, int b) {
        if (b !=7) {
            // TODO Auto-generated method stub
            return a + b;
        } else {
            return 0;
        }
    }
}
```

Finished after 0.01 seconds

Runs: 1/1    Errors: 0    Failures: 1

Operations.AdderTest [Runner: JUnit 4] (0.000 s)

- test (0.000 s)

Failure Trace

```
java.lang.AssertionError: can't add expected:<10> but was:<0>
    at Operations.AdderTest.test(AdderTest.java:30)
```

```
Multiplier.java MultiplierTest.java Adder.java AdderTest.java
```

```
/**
package Operations;

import static org.junit.Assert.*;

/**
 * @author Oded
 */
public class AdderTest {

    /**
     * @throws java.lang.Exception
     */
    @Before
    public void setUp() throws Exception {

    }

    @Test
    public void test() {
        Adder a = new Adder();
        int i = 1;
        while (i < 100) {
            ++i;
            assertEquals("can't add", i+i, a.oper(3, i));
        }
    }
}
```

Don't like these changes?

**Revert back to previous**

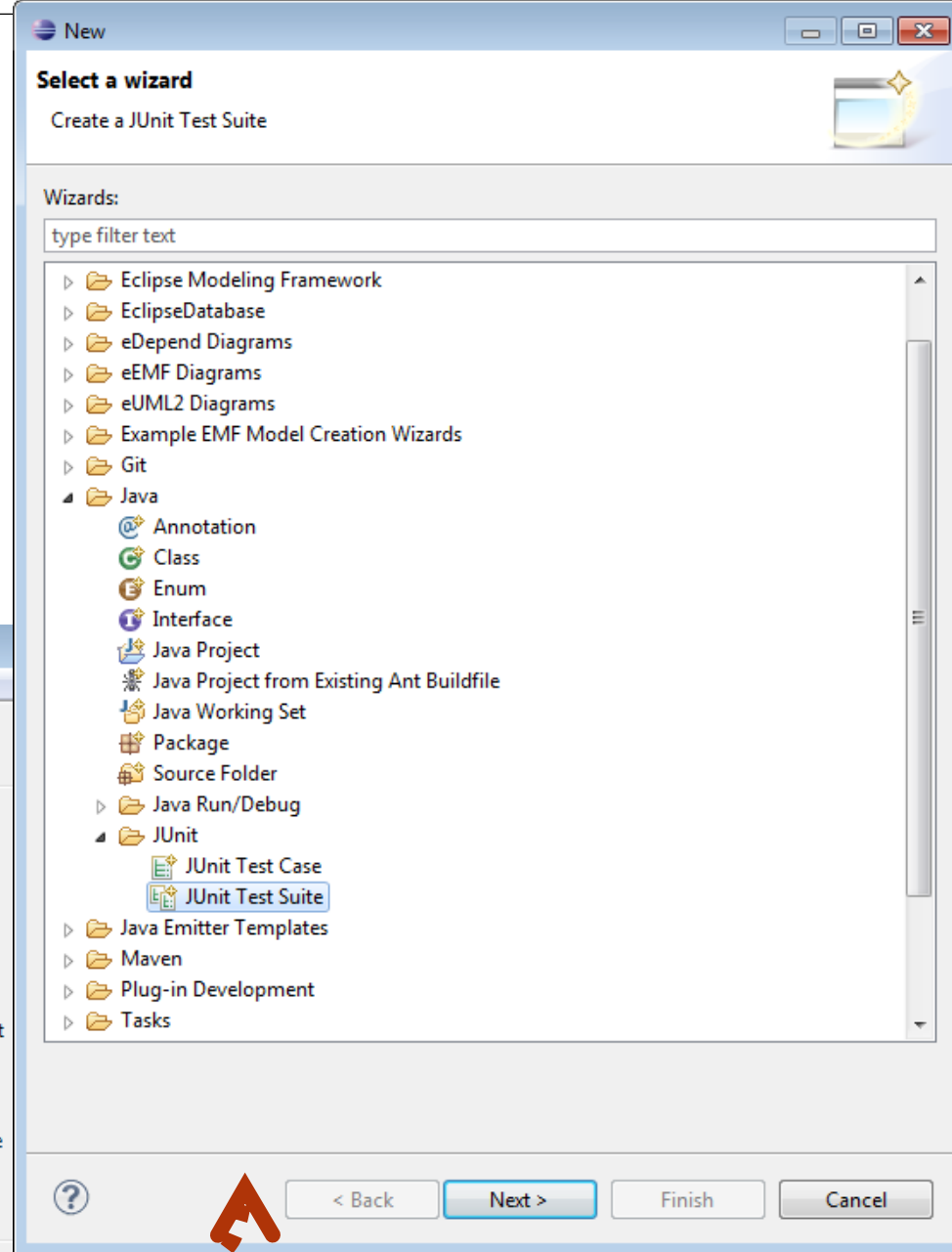
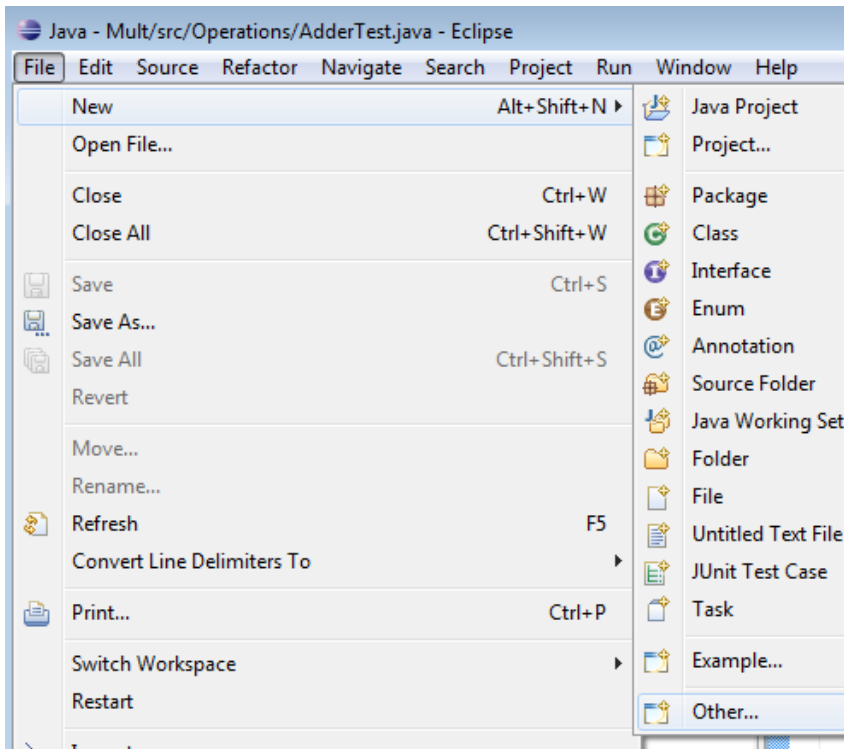


# **More than One Test. Test Suite**

# Test suite

If one has more than one test they can be combined into a test suite.

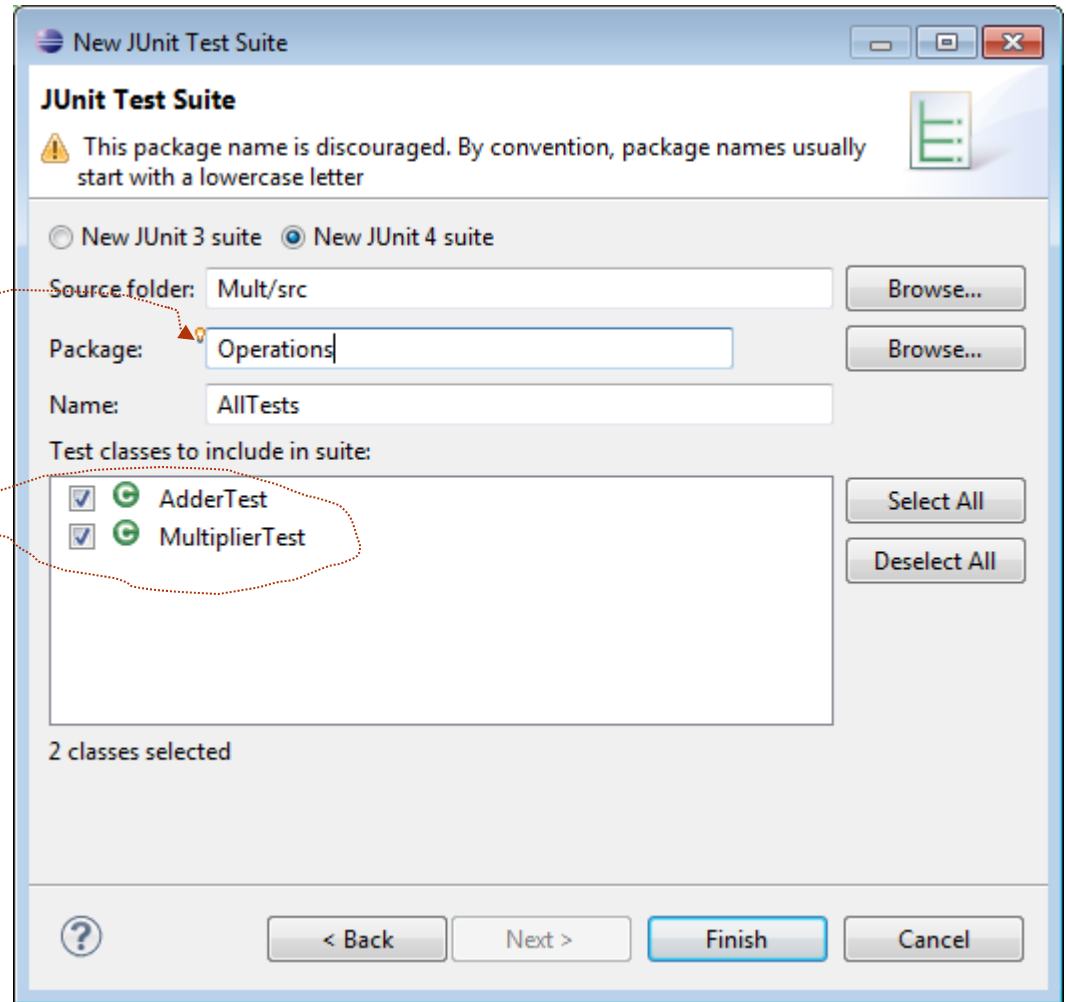
Test suite generated automatically



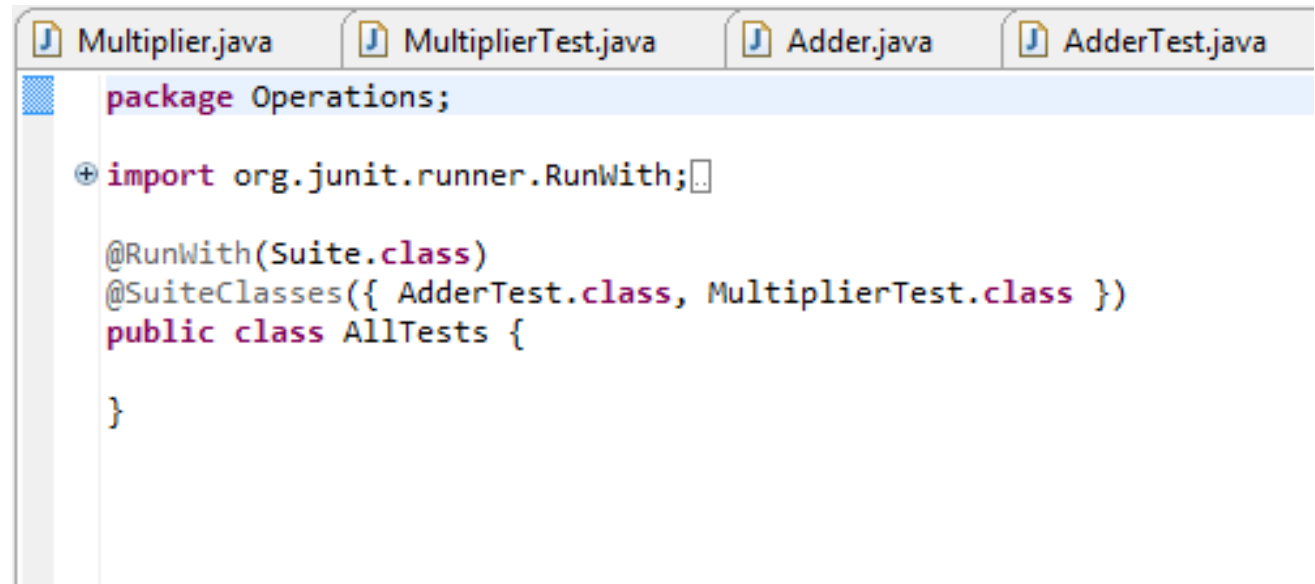
# Test Suite

Fill in Package

Choose tests



# Test Suite Code



```
Multiplier.java MultiplierTest.java Adder.java AdderTest.java
package Operations;

import org.junit.runner.RunWith;

@RunWith(Suite.class)
@SuiteClasses({ AdderTest.class, MultiplierTest.class })
public class AllTests {

}
```

Note that it is possible to run all the test without the test suite.

# Things to Notice

We are writing classes

We did not bother with the main method

We do not have any of these annoying print

# More About Java

# Types

Java programming language is statically-typed

All variables must first be declared before they can be used.  
(anyone know a language that is not)

Java's types

- PrimitiveType
- ReferenceType
- null Type

# PrimitiveType

- Predefined
- Their names are reserved keywords

Primitive data types supported by Java:

- **byte** - 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive)
- **short** - 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive)
- **int** - 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive)
- **long** - The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive)
- **float** - single-precision 32-bit IEEE 754 floating point
- **double** - double-precision 64-bit IEEE 754 floating point
- **boolean** - only two possible values: true and false
- **char** - is a single 16-bit Unicode character



# ReferenceType

```
public class Adder { ...  
}
```

```
Adder a = new Adder();
```

“a” is not an object of type Adder,  
It is a reference to type Adder

Other types that are ReferenceType: interface and array

# Parameter Passing

- In Java parameters are always passed to a method by value.
- Objects are not passed to methods
- So what is passed on instead of the object?
- Instead of the object a reference is passed on to the method.

What does this mean?

Lets let Eclipse and JUnit help us understand.