

## More Testing on Steriods

EECS 4315

`wiki.eecs.yorku.ca/course/4315/`

# Testing a main method

```
@Test
public void test() {
    // command line arguments
    String[] args = {};
    // input given by the user via the keyboard
    String user = "...";

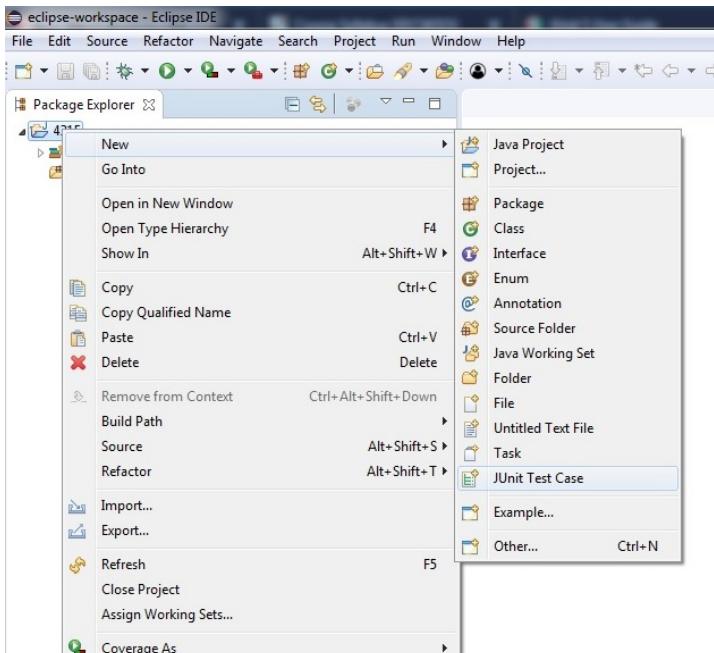
    // set up input and output
    ByteArrayInputStream input =
        new ByteArrayInputStream(user.getBytes());
    System.setIn(input);
    ByteArrayOutputStream output =
        new ByteArrayOutputStream();
    PrintStream stream = new PrintStream(output);
    System.setOut(stream);
}
```

## Testing a main method (continued)

```
// call the main method
ClassName.main(args);

// verify the output
String expected = "...";
String actual = output.toString();
Assertions.assertEquals(expected, actual);
}
```


# Creating a JUnit test case in eclipse



# Creating a JUnit test case in eclipse

New JUnit Test Case

**JUnit Test Case**

 The use of the default package is discouraged.

New JUnit 3 test    New JUnit 4 test    New JUnit Jupiter test

Source folder:

Package:  (default)

Name:

Superclass:

Which method stubs would you like to create?

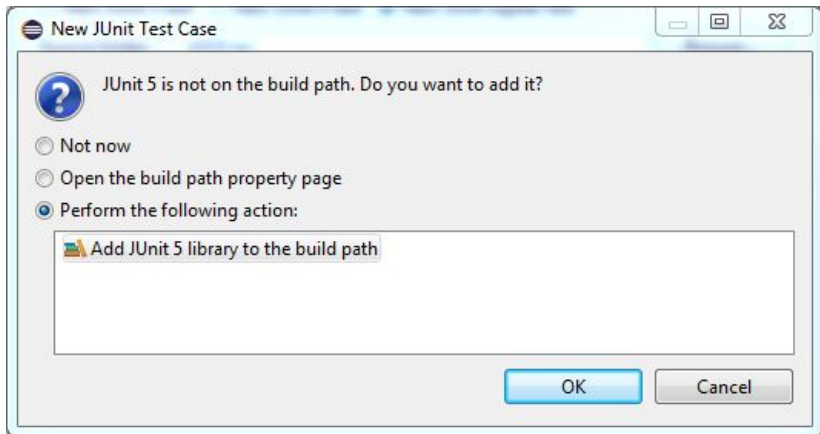
setUpBeforeClass()    tearDownAfterClass()  
 setUp()    tearDown()  
 constructor

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

Generate comments

Class under test:

# Creating a JUnit test case in eclipse



## Creating a JUnit test case in eclipse

```
MyTest.java ✖
1+ import static org.junit.jupiter.api.Assertions.*;
4
5 class MyTest {
6
7-     @Test
8     void test() {
9         fail("Not yet implemented");
10    }
11
12 }
13
```

# Running a JUnit test case in eclipse

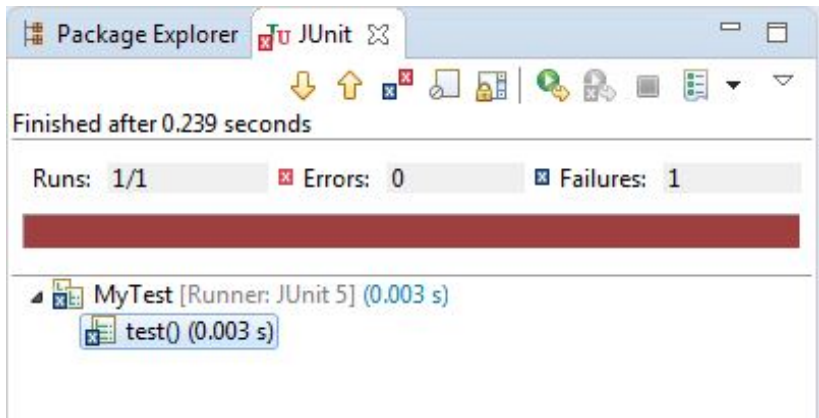
The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays a project named '4315' with a source folder 'src' containing a package '(default package)' and a class 'MyTest.java'. The 'JUnit 5' folder is also visible. The main editor window shows the code for 'MyTest.java':

```
1 import static org.junit.jupiter.api.Assertions.*;
4
5 class MyTest {
6
7     @Test
8     void test() {
9         fail("Not yet implemented");
10    }
```

A context menu is open over the 'MyTest.java' file, listing various actions such as 'New', 'Open', 'Copy', 'Paste', 'Delete', 'Run As', and 'Debug As'. The 'Run As' option is highlighted, and a sub-menu is visible, showing 'JUnit 5' as the selected runner, with the keyboard shortcut 'Alt+Shift+X, T'. Other options in the sub-menu include 'Run Configurations...'. The bottom status bar shows '1 JUnit Test' and 'Alt+Shift+X, T'.



# Running a JUnit test case in eclipse



The screenshot shows the Eclipse IDE's Package Explorer and JUnit runner interface. The Package Explorer is on the left, and the JUnit runner is on the right. The JUnit runner shows the following information:

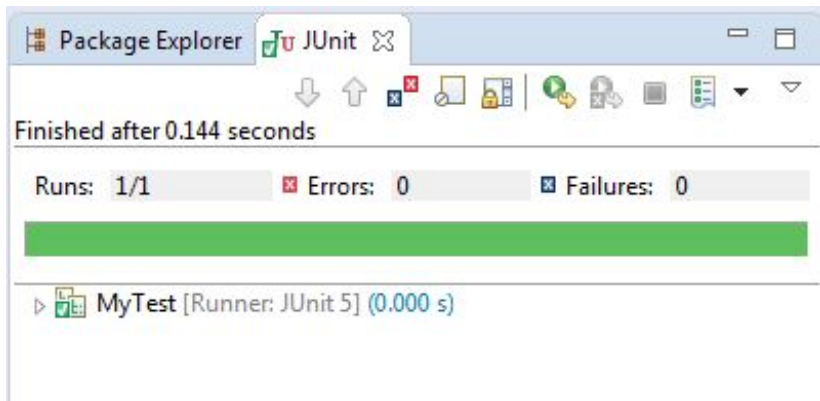
- Package Explorer: JUnit
- Finished after 0.239 seconds
- Runs: 1/1
- Errors: 0
- Failures: 1
- MyTest [Runner: JUnit 5] (0.003 s)
  - test() (0.003 s)

The test results are displayed in a tree view. The 'MyTest' folder is expanded, showing the 'test()' method. The 'test()' method is highlighted with a blue box, indicating it is the current test case. The 'test()' method is marked with a red 'X' icon, indicating a failure. The 'MyTest' folder is marked with a red 'X' icon, indicating a failure. The 'Runs: 1/1' status indicates that one test case was executed. The 'Errors: 0' status indicates that no errors occurred during the test run. The 'Failures: 1' status indicates that one test case failed. The 'MyTest [Runner: JUnit 5] (0.003 s)' status indicates that the test case 'MyTest' was executed using JUnit 5 and took 0.003 seconds to run. The 'test() (0.003 s)' status indicates that the 'test()' method was executed and took 0.003 seconds to run.

## Running a JUnit test case in eclipse

```
*MyTest.java ✖  
1+ import static org.junit.jupiter.api.Assertions.*;   
4  
5 class MyTest {  
6  
7- @Test  
8 void test() {  
9 //fail("Not yet implemented");  
10 }  
11  
12 }  
13
```

# Running a JUnit test case in eclipse



The screenshot shows the Eclipse IDE's Package Explorer and JUnit runner interface. The Package Explorer is on the left, and the JUnit runner is on the right. The JUnit runner shows the following information:

- Package Explorer: JUnit
- JUnit Runner: Finished after 0.144 seconds
- Runs: 1/1
- Errors: 0
- Failures: 0
- Test Case: MyTest [Runner: JUnit 5] (0.000 s)

The JUnit runner interface includes a toolbar with icons for navigation and actions. Below the toolbar, the text "Finished after 0.144 seconds" is displayed. The status bar shows "Runs: 1/1", "Errors: 0", and "Failures: 0". A green progress bar is visible below the status bar. The test case "MyTest" is listed with a green checkmark icon, indicating a successful run.

Write a JUnit test case to test the class `Boolean`, whose API can be found [here](#). Its JAR can be found [here](#).

## Question

What can we test about the constructor?

# Test the constructor

## Question

What can we test about the constructor?

## Answer

That the created object is not null.

# Test the booleanValue method

## Question

What can we test about the booleanValue method?

# Test the booleanValue method

## Question

What can we test about the booleanValue method?

## Answer

Check if it returns the correct value.



# Test the constant TRUE

## Question

What can we test about the constant TRUE?

# Test the constant TRUE

## Question

What can we test about the constant TRUE?

## Answer

Check if it is not null and has the correct value.

# Test the compareTo method

## Question

What can we test about the compareTo method?

# Test the compareTo method

## Question

What can we test about the compareTo method?

## Answer

- 1 Check if it returns a correct value.
- 2 Check if it throws an `IllegalArgumentException` if the argument is `null`.

# Test the compareTo method

## Question

How many “inputs” does the `compareTo` method have?

# Test the compareTo method

## Question

How many “inputs” does the `compareTo` method have?

## Answer

Two: `one.compareTo(two)`

# Test the compareTo method

## Question

How many “inputs” does the `compareTo` method have?

## Answer

Two: `one.compareTo(two)`

## Question

How many combinations of “inputs” for the `compareTo` method do we have to check?

# Test the compareTo method

## Question

How many “inputs” does the `compareTo` method have?

## Answer

Two: `one.compareTo(two)`

## Question

How many combinations of “inputs” for the `compareTo` method do we have to check?

## Answer

Four.



# Test the compareTo method

```
@Test
public void testCompareTo() {
    Boolean FALSE = new Boolean(false);
    ...
    ... Boolean.TRUE.compareTo(FALSE) ...
}
```

## Question

Should we check if the result is 1?

# Test the compareTo method

```
@Test
public void testCompareTo() {
    Boolean FALSE = new Boolean(false);
    ...
    ... Boolean.TRUE.compareTo(FALSE) ...
}
```

## Question

Should we check if the result is 1?

## Answer

No, we should check if the result is greater than zero.

# Test the compareTo method

## Question

How many “inputs” does `compareTo(null)` have?

# Test the compareTo method

## Question

How many “inputs” does `compareTo(null)` have?

## Answer

One.

# Test the compareTo method

## Question

How many “inputs” does `compareTo(null)` have?

## Answer

One.

## Question

How many combinations of “inputs” for `compareTo(null)` do we have to check?

# Test the compareTo method

## Question

How many “inputs” does `compareTo(null)` have?

## Answer

One.

## Question

How many combinations of “inputs” for `compareTo(null)` do we have to check?

## Answer

Two.

# Test the equals method

## Question

Do we have to test the `equals` method?

# Test the equals method

## Question

Do we have to test the `equals` method?

## Answer

No, since it is not part of the API of the `Boolean` class.



```
@Test
public void test() {
    try {
        call of constructor or method;
    } catch (Exception e) {
        Assertions.fail("Exception was thrown");
    }
}
```

## Question

Do we have to test whether each constructor and method does not throw any exceptions?

```
@Test
public void test() {
    try {
        call of constructor or method;
    } catch (Exception e) {
        Assertions.fail("Exception was thrown");
    }
}
```

## Question

Do we have to test whether each constructor and method does not throw any exceptions?

## Answer

No. If a constructor or method throws an exception, the test case will fail and the exception will be reported.

```
@Test
public void test() {
    Boolean value = new Boolean(true);
    Assertions.assertNotNull(value, "...");
    value = false;
    Assertions.assertFalse(value, "...");
    value = true;
    Assertions.assertTrue(value, "...");
}
```

## Question

Which class is tested, `java.lang.Boolean` or `lab.Boolean`?

```
@Test
public void test() {
    Boolean value = new Boolean(true);
    Assertions.assertNotNull(value, "...");
    value = false;
    Assertions.assertFalse(value, "...");
    value = true;
    Assertions.assertTrue(value, "...");
}
```

## Question

Which class is tested, `java.lang.Boolean` or `lab.Boolean`?

## Answer

`java.lang.Boolean`.