

Agile Software Development

Produced
by

Dr. Siobhán Drohan (sdrohan@wit.ie)

Eamonn de Leastar (edeleastar@wit.ie)



First JUnit Tests (JUnit 3)

JUnit

What is JUnit?



JUnit:

- is a Unit Testing Framework for Java.
- enables you to write and run repeatable tests.
- is used to Unit Test a small piece of code.
- When following TDD, developers should write and execute the JUnit tests before writing any code.

JUnit Versions

- **JUnit 3** (<http://junit.sourceforge.net/junit3.8.1/>)
- **JUnit 4** (<http://junit.org/junit4/>)
 - Version we will mainly use is 4.12
- **JUnit 5** (<http://junit.org/junit5/>)
 - First general availability release published on September 10, 2017. Bug fix version released on October 3, 2017.
 - Not supported directly in Eclipse IDE (you can use it via Maven, which we will show you in future weeks)

JUnit Version 3

- *As conventions differ between the versions, it is important to be able to use Version 3 and 4 (at least) and 5 (desirable).*
- In Version 3:

1. Test class must extend **TestCase**.
2. setUp/tearDown methods are overridden from **TestCase** (note that this is optional).
3. Test methods must begin with “test” word.

1. Test Class must extend **TestCase**

unit3.8.1/javadoc/index.html



[Overview](#) [Package](#) **[Class](#)** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

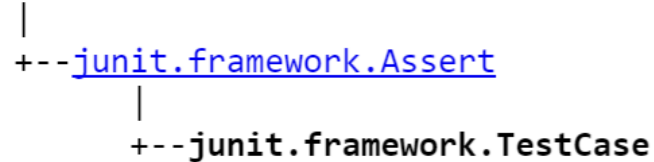
[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

junit.framework

Class TestCase

java.lang.Object



All Implemented Interfaces:

[Test](#)

Direct Known Subclasses:

[ActiveTestTest](#), [ActiveTestTest.SuccessTest](#), [AssertTest](#), [BaseTestRunnerTest](#), [ComparisonFailureTest](#), [DoublePrecisionAssertTest](#), [ExceptionTestCase](#), [ExceptionTestCaseTest](#), [ExtensionTest](#), [Failure](#), [MoneyTest](#), [NoArgTestCaseTest](#), [NoTestCases](#), [NotPublicTestCase](#), [NotVoidTestCase](#), [OneTestCase](#), [RepeatedTestTest](#), [RepeatedTestTest.SuccessTest](#), [SimpleTest](#), [SimpleTestCollectorTest](#), [SorterTest](#), [StackFilterTest](#), [Success](#), [SuiteTest](#), [TestCaseClassLoaderTest](#), [TestCaseTest](#), [TestCaseTest.TornDown](#), [TestImplementorTest](#), [TestListenerTest](#), [TextFeedbackTest](#), [TextRunnerTest](#), [VectorTest](#), [WasRun](#)

```
public abstract class TestCase
```

```
extends Assert
```

```
implements Test
```

A test case defines the fixture to run multiple tests. To define a test case

- 1) implement a subclass of TestCase
- 2) define instance variables that store the state of the fixture
- 3) initialize the fixture state by overriding setUp
- 4) clean-up after a test by overriding tearDown.

1. Test Class must extend **TestCase**

unit3.8.1/javadoc/index.html

[Overview](#) [Package](#) **[Class](#)** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

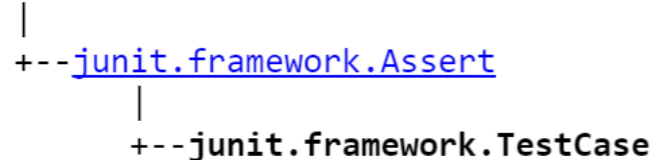
[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

junit.framework

Class TestCase

java.lang.Object



All Implemented Interfaces:

[Test](#)

Direct Known Subclasses:

[ActiveTestTest](#), [ActiveTestTest.SuccessTest](#), [AssertTest](#), [BaseTestRunnerTest](#), [ComparisonFailureTest](#), [DoublePrecisionAssertTest](#), [ExceptionTestCase](#), [ExceptionTestCaseTest](#), [ExtensionTest](#), [Failure](#), [MoneyTest](#), [NoArgTestCaseTest](#), [NoTestCases](#), [NotPublicTestCase](#), [NotVoidTestCase](#), [OneTestCase](#), [RepeatedTestTest](#), [RepeatedTestTest.SuccessTest](#), [SimpleTest](#), [SimpleTestCollectorTest](#), [SorterTest](#), [StackFilterTest](#), [Success](#), [SuiteTest](#), [TestCaseClassLoaderTest](#), [TestCaseTest](#), [TestCaseTest.TornDown](#), [TestImplementorTest](#), [TestListenerTest](#), [TextFeedbackTest](#), [TextRunnerTest](#), [VectorTest](#), [WasRun](#)

```
public abstract class TestCase
```

```
extends Assert
```

```
implements Test
```

A test case defines the fixture to run multiple tests. To define a test case

- 1) implement a subclass of TestCase
- 2) define instance variables that store the state of the fixture
- 3) initialize the fixture state by overriding setUp
- 4) clean-up after a test by overriding tearDown.

```
import junit.framework.TestCase;

public class TestLargest extends TestCase
{
    //JUnit testing code omitted
}
```

2. setUp/tearDown methods are overridden from **TestCase** (note that this is optional).

TestCase (JUnit API)

junit.sourceforge.net/junit3.8.1/javadoc/index.html

[All Classes](#)

Packages

- [junit.awtui](#)
- [junit.extensions](#)
- [junit.framework](#)
- [junit.runner](#)
- [junit.samples](#)

[junit.framework](#)

Interfaces

- [Protectable](#)
- [Test](#)
- [TestListener](#)



Classes

- [Assert](#)
- [TestCase](#)
- [TestFailure](#)
- [TestResult](#)
- [TestSuite](#)

Errors

- [AssertionFailedError](#)
- [ComparisonFailure](#)

Method Summary

int	countTestCases()	Counts the number of test cases executed by run(TestResult result).
protected TestResult	createResult()	Creates a default TestResult object
java.lang.String	getName()	Gets the name of a TestCase
TestResult	run()	A convenience method to run this test, collecting the results with a default TestResult object.
void	run(TestResult result)	Runs the test case and collects the results in TestResult.
void	runBare()	Runs the bare test sequence.
protected void	runTest()	Override to run the test and assert its state.
void	setName(java.lang.String name)	Sets the name of a TestCase
protected void	setUp()	Sets up the fixture, for example, open a network connection. 
protected void	tearDown()	Tears down the fixture, for example, close a network connection. 
java.lang.String	toString()	Returns a string representation of the test case

3. Test methods must begin with “test” word.

```
import junit.framework.TestCase;

public class TestLargest extends TestCase
{
    public TestLargest (String name)
    {
        super(name);
    }

    public void testOrder () ←
    {
        int[] arr = new int[3];
        arr[0] = 8;
        arr[1] = 9;
        arr[2] = 7;
        assertEquals(9, Largest.largest(arr));
    }
}
```

Let's look at Assertions now...and then we will look at a JUnit testing a simple program.

```
import junit.framework.TestCase;

public class TestLargest extends TestCase
{
    public TestLargest (String name)
    {
        super(name);
    }

    public void testOrder ()
    {
        int[] arr = new int[3];
        arr[0] = 8;
        arr[1] = 9;
        arr[2] = 7;
        assertEquals(9, Largest.largest(arr));
    }
}
```



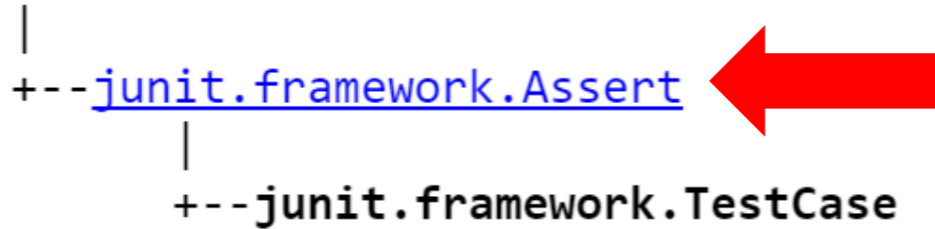
Assertions

- To check if code is behaving as you expect
 - use an *assertion* *i.e.* a simple method call that verifies that something is true.

junit.framework

Class TestCase

java.lang.Object



All Implemented Interfaces:

[Test](#)

Direct Known Subclasses:

[ActiveTestTest](#), [ActiveTestTest.SuccessTest](#), [AssertTest](#), [BaseTestRunnerTest](#), [ExceptionTestCase](#), [ExceptionTestCaseTest](#), [ExtensionTest](#), [Failure](#), [MoneyTest](#), [NotVoidTestCase](#), [OneTestCase](#), [RepeatedTestTest](#), [RepeatedTestTest.SuccessTest](#), [StackFilterTest](#), [Success](#), [SuiteTest](#), [TestCaseClassLoaderTest](#), [TestCaseTest](#), [TextFeedbackTest](#), [TextRunnerTest](#), [VectorTest](#), [WasRun](#)

```
public abstract class TestCase
extends Assert
implements Test
```

A test case defines the fixture to run multiple tests. To define a test case

- 1) implement a subclass of TestCase
- 2) define instance variables that store the state of the fixture
- 3) initialize the fixture state by overriding setUp
- 4) clean-up after a test by overriding tearDown.

junit.framework

Class Assert

java.lang.Object

|

+-- junit.framework.Assert

Direct Known Subclasses:

[ClassLoaderTest](#), [LoadedFromJar](#), [TestCase](#), [TestDecorator](#)

```
public class Assert
```

```
extends java.lang.Object
```

A set of assert methods. Messages are only displayed when an assert fails.



Some of the many “assertion” methods in the Assert class...

static void	<u>assertEquals</u> (java.lang.String expected, java.lang.String actual) Asserts that two Strings are equal.
static void	<u>assertEquals</u> (java.lang.String message, java.lang.String expected, java.lang.String actual) Asserts that two Strings are equal.
static void	<u>assertFalse</u> (boolean condition) Asserts that a condition is false.
static void	<u>assertFalse</u> (java.lang.String message, boolean condition) Asserts that a condition is false.
static void	<u>assertNotNull</u> (java.lang.Object object) Asserts that an object isn't null.
static void	<u>assertNotNull</u> (java.lang.String message, java.lang.Object object) Asserts that an object isn't null.
static void	<u>assertNotSame</u> (java.lang.Object expected, java.lang.Object actual) Asserts that two objects do not refer to the same object.
static void	<u>assertNotSame</u> (java.lang.String message, java.lang.Object expected, java.lang.Object actual) Asserts that two objects do not refer to the same object.
static void	<u>assertNull</u> (java.lang.Object object) Asserts that an object is null.
static void	<u>assertNull</u> (java.lang.String message, java.lang.Object object) Asserts that an object is null.
static void	<u>assertSame</u> (java.lang.Object expected, java.lang.Object actual) Asserts that two objects refer to the same object.

Using Asserts

You could use this assert to check all sorts of things, including whether numbers are equal to each other.

```
int a = 2;  
//...  
assertTrue (a == 2);
```

Using Asserts

You could use this assert to check all sorts of things, including whether numbers are equal to each other.

```
int a = 2;  
//...  
assertTrue (a == 2);
```

To check that two integers are equal, a method that takes two integer parameters might be more useful.

```
public void assertEquals (int a, int b)  
{  
    assertTrue(a == b);  
}
```


Using Asserts

You could use this assert to check all sorts of things, including whether numbers are equal to each other.

```
int a = 2;  
//...  
assertTrue (a == 2);
```

To check that two integers are equal, a method that takes two integer parameters might be more useful.

```
public void assertEquals (int a, int b)  
{  
    assertTrue(a == b);  
}
```

We can now write the first test a little more expressively:

```
int a = 2;  
  
assertEquals (2, a);
```

JUnit Example

Testing code to return the largest number in a Primitive Array.

Planning Tests

- Method to test: A static method designed to find the largest number in a list of numbers.
- The following tests would seem to make sense:
 - **[7, 8, 9]** → **9**
 - **[8, 9, 7]** → **9**
 - **[9, 7, 8]** → **9**

```
public static int largest (int[] list)
{
    ...
}
```

(**supplied test data** → **expected result**)

More Test Data + First Implementation

- Already have this data:

[7, 8, 9] → 9

[8, 9, 7] → 9

[9, 7, 8] → 9

- What about this set:

[7, 9, 8, 9] → 9

[1] → 1

[-9, -8, -7] → -7

```
public static int largest (int[] list)
{
    int index, max = Integer.MAX_VALUE;

    for (index = 0; index < list.length - 1; index++)
    {
        if (list[index] > max)
        {
            max = list[index];
        }
    }

    return max;
}
```

(supplied test data → expected result)

Writing the TestCase

- This is a **TestCase** called TestLargest.
- It uses the following test data:
 [8, 9, 7] → 9
- It has one Unit Test (testOrder) - to verify the behaviour of the largest method.

```
import junit.framework.TestCase;

public class TestLargest extends TestCase
{
    public TestLargest (String name)
    {
        super(name);
    }

    public void testOrder () ←
    {
        int[] arr = new int[3];
        arr[0] = 8;
        arr[1] = 9;
        arr[2] = 7;
        assertEquals(9, Largest.largest(arr)); ←
    }
}
```

Running the TestCase

The screenshot shows the JUnit runner interface in an IDE. At the top, it says "Finished after 0.008 seconds". Below that, it displays the execution statistics: "Runs: 1/1", "Errors: 0", and "Failures: 1". A progress bar is shown below these statistics, with a red segment indicating the failed test. The test hierarchy is listed as follows:

- TestLargest [Runner: JUnit 3] (0.000 s)
 - testOrder (0.000 s)

At the bottom, the "Failure Trace" section shows the following error message:

```
junit.framework.AssertionFailedError: expected:<9> but was:<2147483647>  
at TestLargest.testOrder(TestLargest.java:42)
```

Status of the Automated Test Execution. One test ran and that one test failed.

Lists the test classes and all the test methods within them.

Lists all the tests that failed, along with assertion errors.

Running the TestCase

The screenshot shows the JUnit runner interface in an IDE. At the top, it says "Finished after 0.008 seconds". Below that, it displays "Runs: 1/1", "Errors: 0", and "Failures: 1". A red progress bar indicates the test failed. The test tree shows "TestLargest [Runner: JUnit 3] (0.000 s)" expanded to show "testOrder (0.000 s)". The "Failure Trace" section shows the following error message:

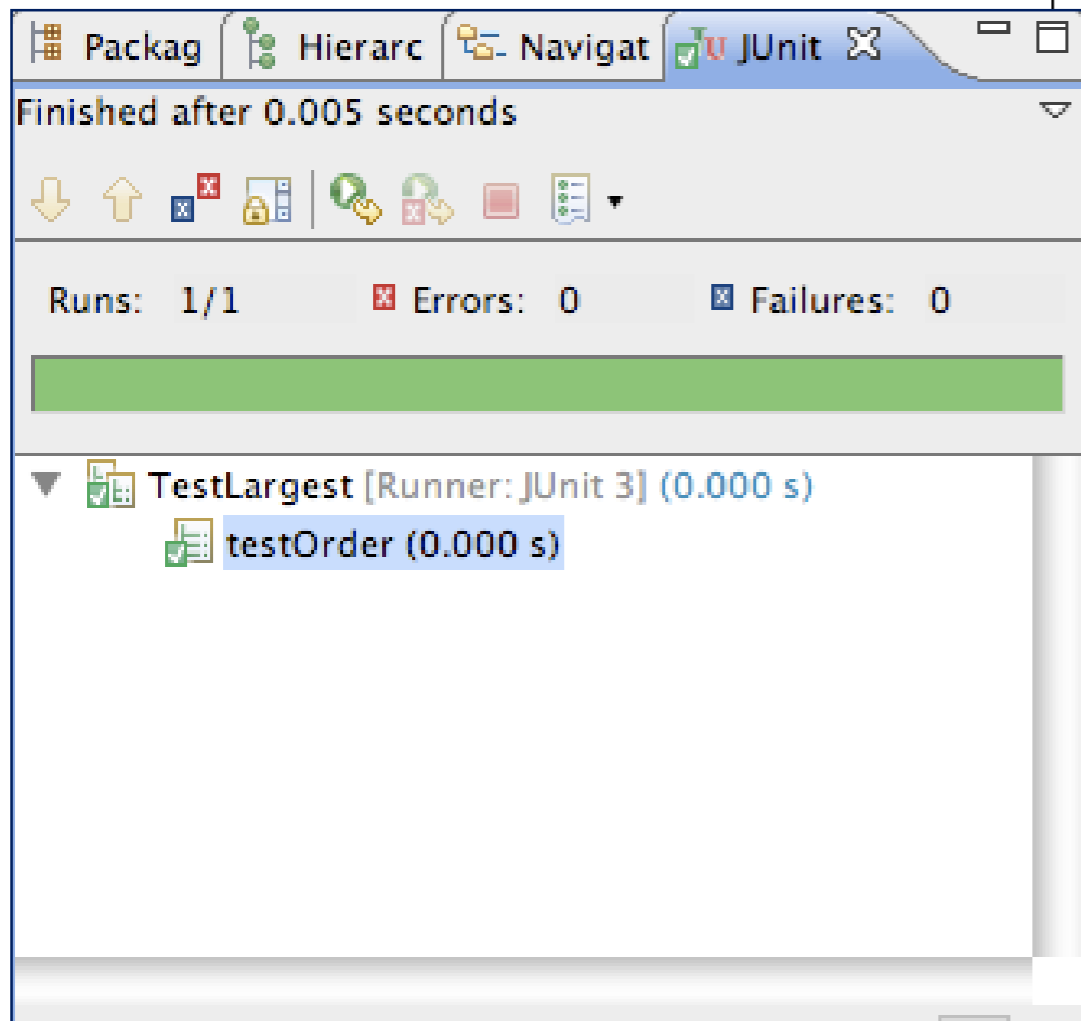
```
junit.framework.AssertionFailedError: expected:<9> but was:<2147483647>  
at TestLargest.testOrder(TestLargest.java:42)
```

A callout box with a pink background and a white arrow pointing to the error message contains the text:

Why did the test return such a huge number instead of 9?
Where could this large number have come from?

Bug

- First line should initialize max to zero, not MAX_VALUE.



```
public static int largest (int[] list)
{
    //int index, max = Integer.MAX_VALUE;
    int index, max = 0;

    for (index = 0; index < list.length - 1; index++)
    {
        if (list[index] > max)
        {
            max = list[index];
        }
    }

    return max;
}
```


Further Tests

- What happens when the largest number appears in different places in the list - first or last, and somewhere in the middle?
 - Bugs most often show up at the “edges”.
 - In this case, edges occur when the largest number is at the start or end of the array that we pass in.
- Aggregate into a single unit test:

```
public void testOrder ()
{
    assertEquals(9, Largest.largest(new int[] { 9, 8, 7 }));
    assertEquals(9, Largest.largest(new int[] { 8, 9, 7 }));
    assertEquals(9, Largest.largest(new int[] { 7, 8, 9 }));
}
```

Failure

Finished after 0.01 seconds

Runs: 1/1 Errors: 0 Failures: 1

testOrder [Runner: JUnit 3] (0.001 s)

Failure Trace

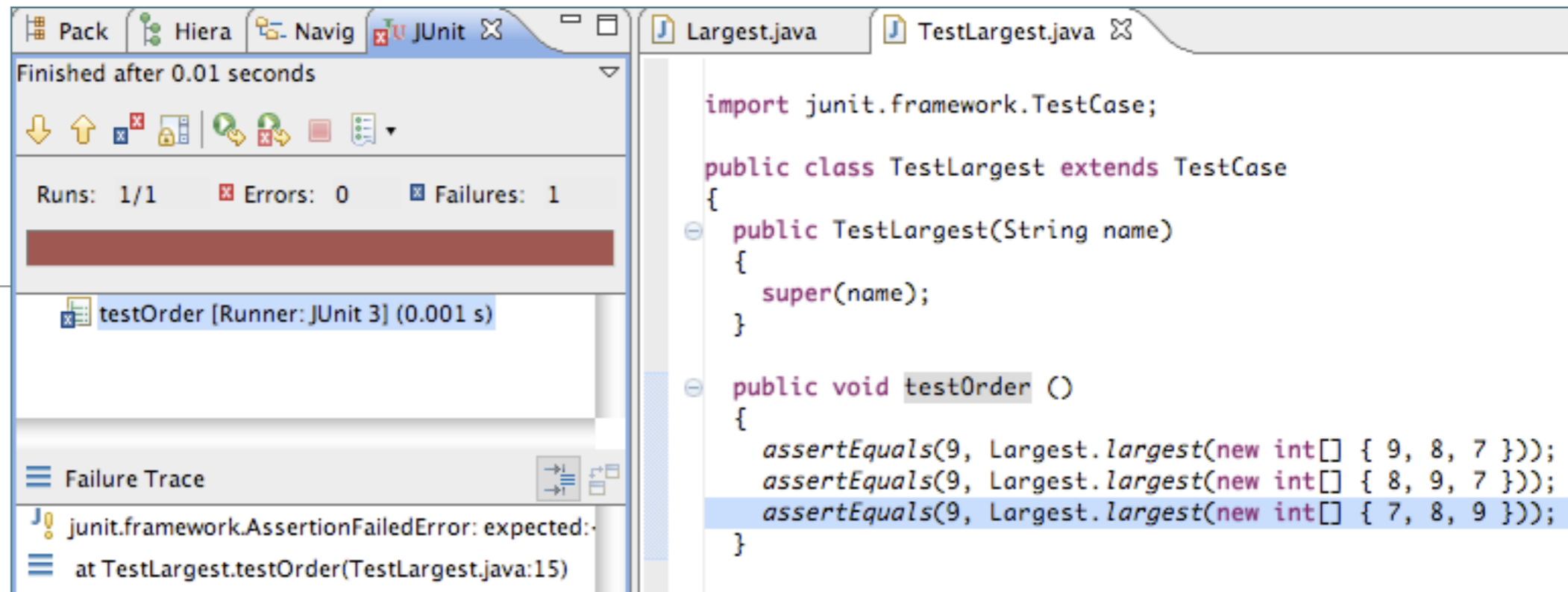
junit.framework.AssertionFailedError: expected:
at TestLargest.testOrder(TestLargest.java:15)

```
import junit.framework.TestCase;

public class TestLargest extends TestCase
{
    public TestLargest(String name)
    {
        super(name);
    }

    public void testOrder ()
    {
        assertEquals(9, Largest.largest(new int[] { 9, 8, 7 }));
        assertEquals(9, Largest.largest(new int[] { 8, 9, 7 }));
        assertEquals(9, Largest.largest(new int[] { 7, 8, 9 }));
    }
}
```

Failure + Fix



The screenshot shows an IDE with two windows. The left window, titled 'JUnit', displays the results of a test run. It indicates that the test finished after 0.01 seconds, with 1 run, 0 errors, and 1 failure. The failure is for the test 'testOrder' [Runner: JUnit 3] (0.001 s). Below this, a 'Failure Trace' shows the error: 'junit.framework.AssertionFailedError: expected:'. The right window shows the source code for 'TestLargest.java'. The code defines a class 'TestLargest' that extends 'TestCase'. It has a constructor 'TestLargest(String name)' and a test method 'testOrder()' which calls 'assertEquals' three times with the value 9 and the result of 'Largest.largest' for different integer arrays: {9, 8, 7}, {8, 9, 7}, and {7, 8, 9}. The third assertion is highlighted in blue, indicating it is the one that failed.

```
import junit.framework.TestCase;

public class TestLargest extends TestCase
{
    public TestLargest(String name)
    {
        super(name);
    }

    public void testOrder ()
    {
        assertEquals(9, Largest.largest(new int[] { 9, 8, 7 }));
        assertEquals(9, Largest.largest(new int[] { 8, 9, 7 }));
        assertEquals(9, Largest.largest(new int[] { 7, 8, 9 }));
    }
}
```

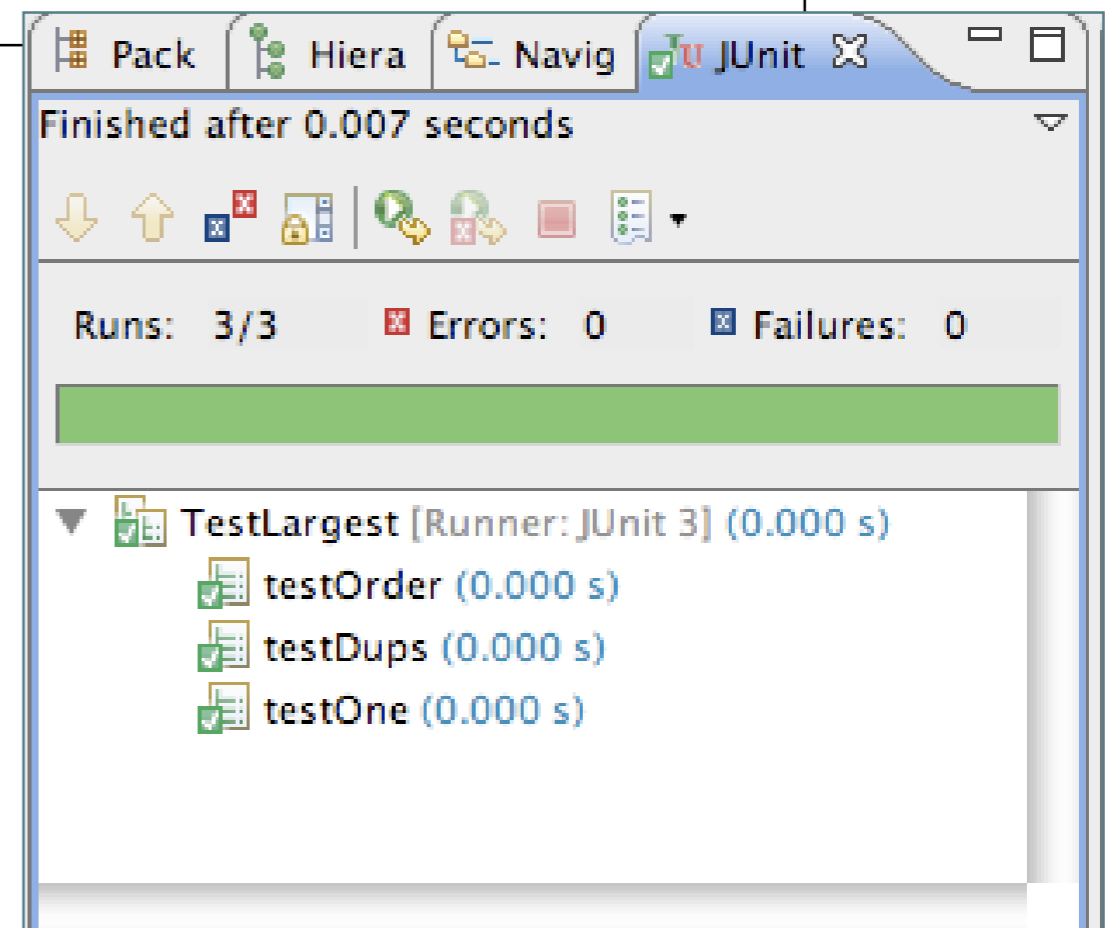
```
public static int largest (int[] list)
{
    int index, max = 0;
    //for (index = 0; index < list.length - 1; index++)
    for (index = 0; index < list.length; index++)
    {
        if (list[index] > max)
        {
            max = list[index];
        }
    }
    return max;
}
```

Further Boundary Conditions

```
public void testDups ()
{
    assertEquals(9, Largest.largest(new int[] { 9, 7, 9, 8 }));
}

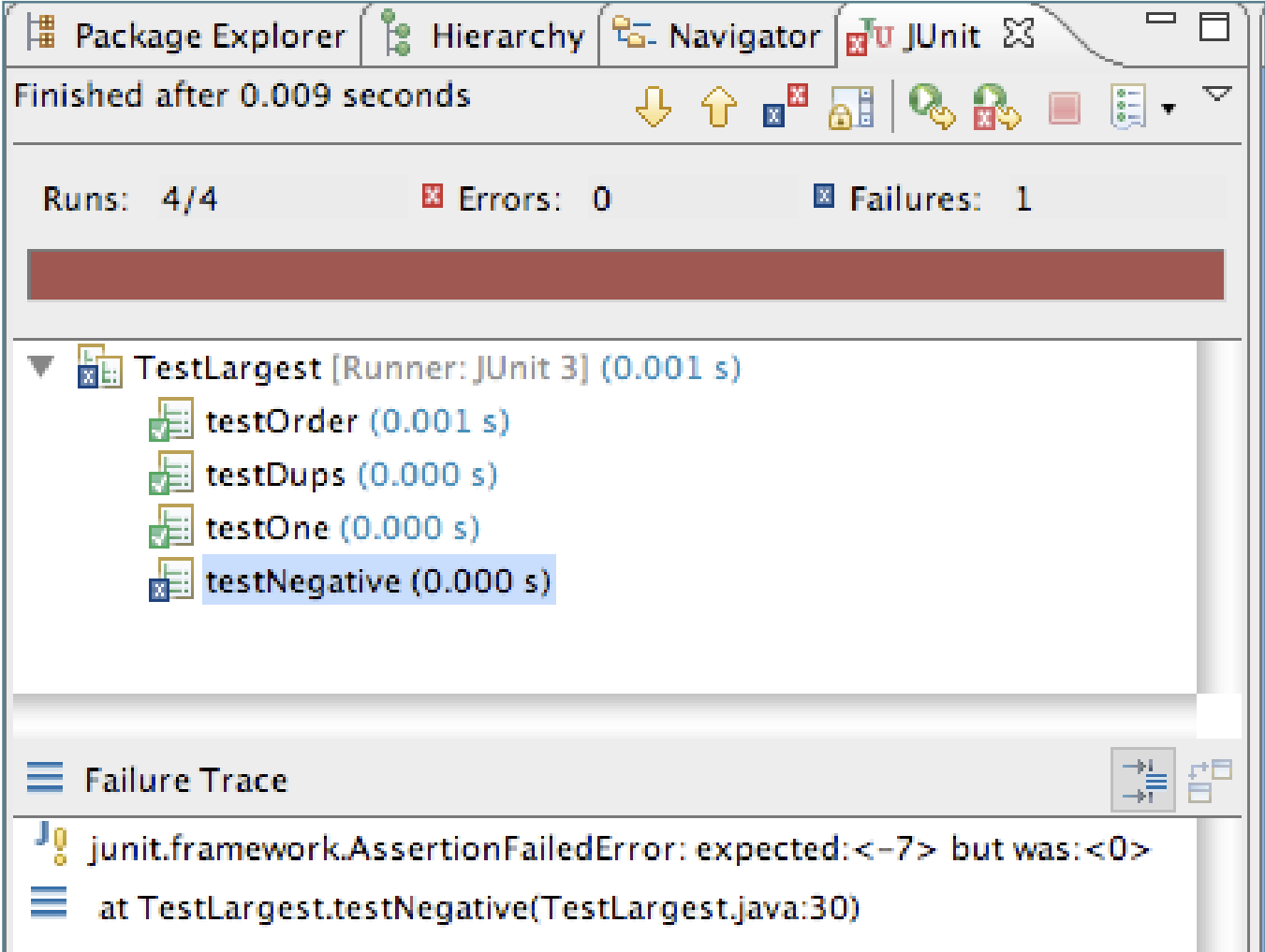
public void testOne ()
{
    assertEquals(1, Largest.largest(new int[] { 1 }));
}
```

- Now exercising multiple tests



Failure on testNegative

```
public void testNegative ()  
{  
    int[] negList = new int[] { -9, -8, -7 };  
    assertEquals(-7, Largest.largest(negList));  
}
```



The screenshot shows the JUnit test runner interface in an IDE. At the top, it indicates "Finished after 0.009 seconds". Below this, a summary bar shows "Runs: 4/4", "Errors: 0", and "Failures: 1". A progress bar is partially filled with red. The test results are listed below, showing four tests: testOrder (0.001 s), testDups (0.000 s), testOne (0.000 s), and testNegative (0.000 s). The testNegative test is highlighted in blue and has a red 'x' icon next to it, indicating a failure. At the bottom, the "Failure Trace" section shows the error message: "junit.framework.AssertionFailedError: expected:<-7> but was:<0>" and the location "at TestLargest.testNegative(TestLargest.java:30)".

fix testNegative

- Choosing 0 to initialize max was a bad idea;
- Should have been MIN VALUE, so as to be less than all negative numbers as well.

```
public static int largest (int[] list)
{
    //int index, max = 0;
    int index, max = Integer.MIN_VALUE;

    for (index = 0; index < list.length; index++)
    {
        if (list[index] > max)
        {
            max = list[index];
        }
    }
    return max;
}
```

Expected Errors?

- If the array is empty, this is considered an error, and an exception should be thrown.

```
public void testEmpty ()
{
    try
    {
        Largest.largest(new int[] {});
        fail("Should have thrown an exception");
    }
    catch (RuntimeException e)
    {
        assertTrue(true);
    }
}
```

```
public static int largest (int[] list)
{
    int index, max = Integer.MIN_VALUE;

    if (list.length == 0)
    {
        throw new RuntimeException("Empty list");
    }
    for (index = 0; index < list.length; index++)
    {
        if (list[index] > max)
        {
            max = list[index];
        }
    }
    return max;
}
```

Some more TDD theory...

TDD – Common Pitfalls (individual programmer)

- Forgetting to run tests frequently
- Writing too many tests at once
- Writing tests that are too large or coarse-grained
- Writing overly trivial tests, for instance omitting assertions
- Writing tests for trivial code, for instance accessors

TDD – Common Pitfalls (teams)

- Partial adoption - only a few developers on the team use TDD.
- Poor maintenance of the test suite - most commonly leading to a test suite with a prohibitively long running time.
- Abandoned test suite (i.e. seldom or never run) - sometimes as a result of poor maintenance, sometimes as a result of team turnover.

TDD – Signs of Use

- ["code coverage"](#) is a common approach to evidencing the use of TDD; while high coverage does not guarantee appropriate use of TDD, coverage below 80% is likely to indicate deficiencies in a team's mastery of TDD.
- [version control](#) logs should show that test code is checked in each time product code is checked in, in roughly comparable amounts.

TDD – Code Coverage – 100% Example

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows a JUnit test run for MiniHashMapTest, which completed successfully after 0.218 seconds with 6/6 runs, 0 errors, and 0 failures. The main editor displays the MiniHashMap.java source code, with the put method highlighted in green to indicate 100% coverage. The bottom-right pane shows the Coverage view for MiniHashMapTest, displaying a table with 100% coverage for all elements.

```
public void put(final K key, final V value) {
    // select bucket
    final int index = getIndex(key.hashCode());
    Entry<K, V> current = values[index];
    if (null == current) {
        // insert if still empty
        values[index] = new Entry<K, V>(key, value);
    } else {
        if (key.equals(current.getKey())) {
            // change value if already in the hash map and stop
            current.setValue(value);
            return;
        }
        while (current.isNotLast()) {
            // search all entries in the bucket
            current = current.getNext();
        }
    }
}
```

Element	Coverage	Covered Lines	Missed Lines	Total Lines
Entry<K, V>	100.0 %	13	0	13
MiniHashMap<K, V>	100.0 %	48	0	48
MiniHashMapTest	100.0 %	89	0	89

TDD – Code Coverage – 85.4% Example

The screenshot displays the Eclipse IDE interface. The main editor shows the `CursorableLinkedList.java` file with the `addAll` method. The code is color-coded, with green highlighting indicating covered lines and red highlighting indicating uncovered lines. The `addAll` method is as follows:

```
public boolean addAll(int index, Collection c) {  
    if(c.isEmpty()) {  
        return false;  
    } else if( size == index || size == 0) {  
        return addAll(c);  
    } else {  
        Listable succ = getListableAt(index);  
        Listable pred = (null == succ) ? null : succ.prev();  
        Iterator it = c.iterator();  
        while(it.hasNext()) {  
            pred = insertListable(pred, succ, it.next());  
        }  
        return true;  
    }  
}
```

The Coverage view at the bottom shows the following table:

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520




TDD – Code Coverage Tool

← → ↻ ⓘ www.eclEmma.org/jacoco/ ☆

EclEmma 3.0.0 Java Code Coverage for Eclipse Install

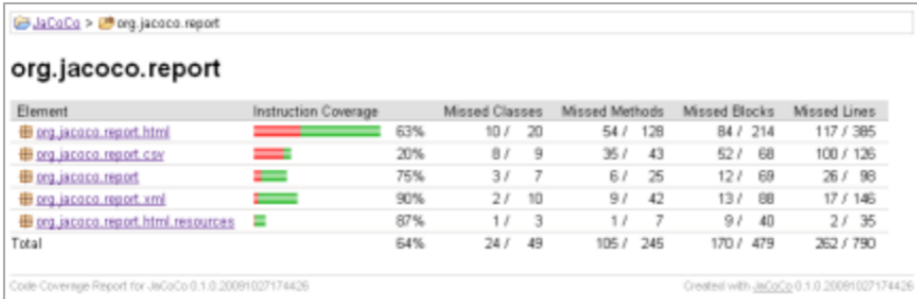
- Overview
- Installation
- User Guide
- Support
- Resources
- Developer Information
- Research
- JaCoCo**
- Change Log
- License
- Contact

GitHub Home



JaCoCo Java Code Coverage Library

JaCoCo is a free code coverage library for Java, which has been created by the EclEmma team based on the lessons learned from using and integration existing libraries for many years.



Element	Instruction Coverage	Missed Classes	Missed Methods	Missed Blocks	Missed Lines
org.jacoco.report.html	63%	10 / 20	54 / 128	84 / 214	117 / 385
org.jacoco.report.csv	20%	9 / 9	35 / 43	52 / 68	100 / 126
org.jacoco.report	75%	3 / 7	6 / 25	12 / 69	26 / 98
org.jacoco.report.xml	90%	2 / 10	9 / 42	13 / 88	17 / 146
org.jacoco.report.html.resources	87%	1 / 3	1 / 7	9 / 40	2 / 35
Total	64%	24 / 49	105 / 245	170 / 479	262 / 790

Code Coverage Report for JaCoCo 0.1.0.20091027174426 Created with JaCoCo 0.1.0.20091027174426

Snapshot Builds

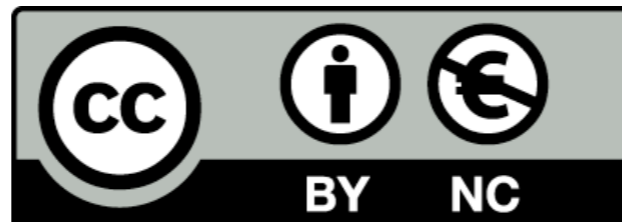
The [master branch](#) of JaCoCo is automatically built and published. Due to the test driven development approach every build is considered fully functional. See [change history](#) for latest features and bug fixes. SonarQube code quality metrics of the current JaCoCo implementation are available on [SonarQube.com](#).

- [Documentation](#)
- [Download \(Maven Repository\)](#)
- [Coverage Report](#)

Release Builds

The official releases builds are available for download below. JaCoCo is also available from the [Maven repository](#).

Download	Source	Release	Size	MD5 Checksum
jacoco-0.7.9.zip	0.7.9	2017/02/05	2.9 MB	9add26ba3689b0b9324284b5231aff51
jacoco-0.7.8.zip	0.7.8	2016/12/10	2.9 MB	dacce9e592038fa0b7307b96e521d797
jacoco-0.7.7.201606060606.zip	0.7.7	2016/06/06	2.9 MB	10f6f6c1fd7152447dd5e38e7c7d5f76



Except where otherwise noted, this content is licensed under a [Creative Commons Attribution-NonCommercial 3.0 License](http://creativecommons.org/licenses/by-nc/3.0/).

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

