Agile Software Development

Produced by

Dr. Siobhán Drohan (sdrohan@wit.ie)

Eamonn de Leastar (edeleastar@wit.ie)

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

First JUnit Tests (JUnit 3)

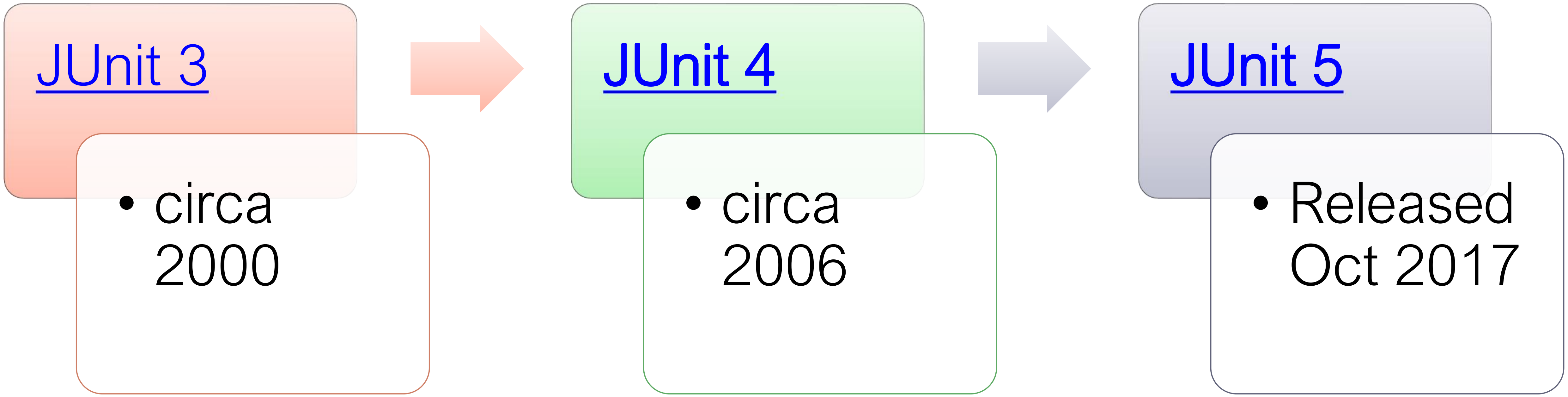
What is JUnit?

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

- circa 2000

JUnit 4

- circa 2006

JUnit 5

- Released Oct 2017

As conventions differ between the versions,
it is important to be able to use/recognise all three versions.

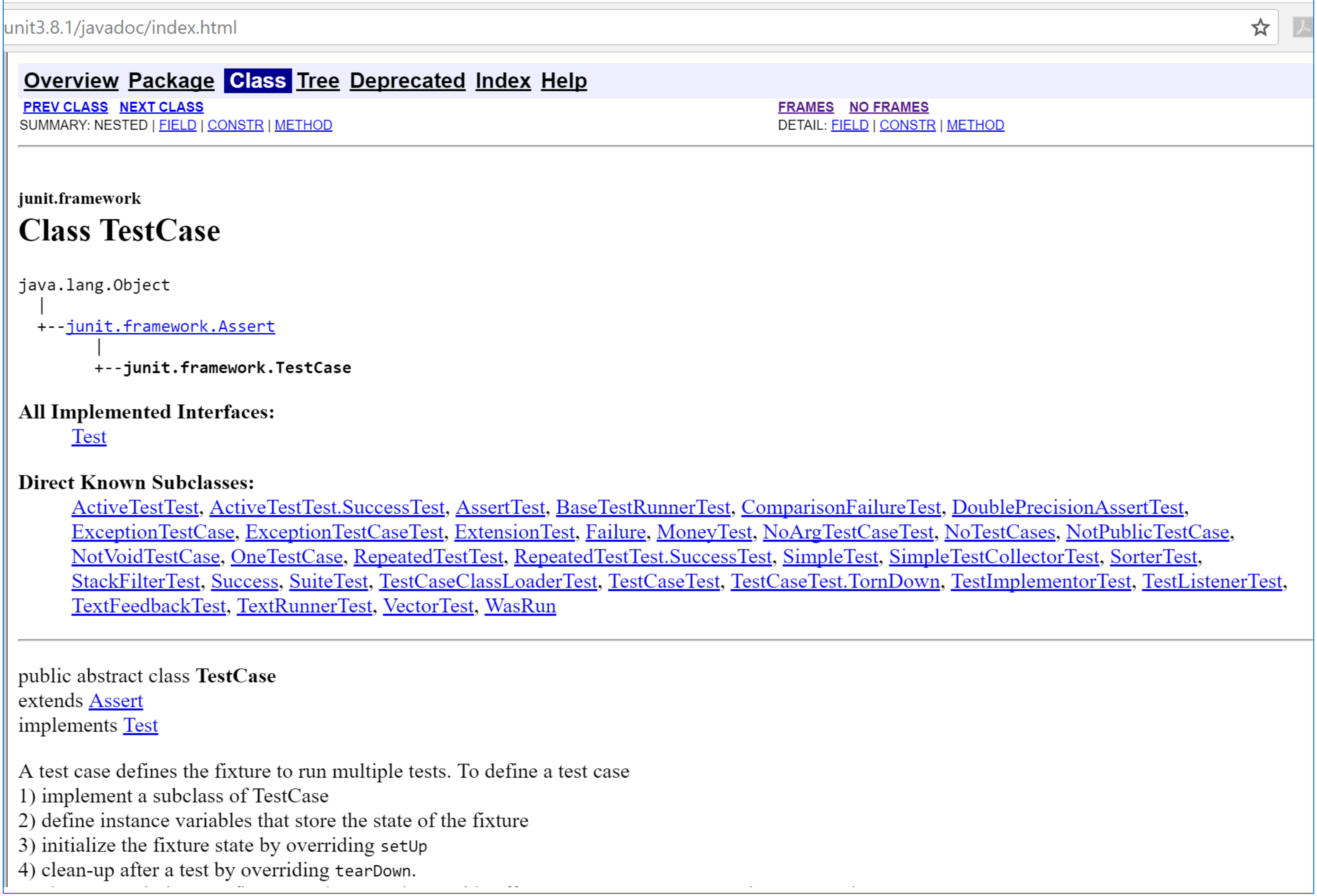
http://wiki.c2.com/?TenYearsOfTestDrivenDevelopment

http://junit.sourceforge.net/doc/testinfected/testing.htm

JUnit Version 3

1. Test class must extend TestCase.
2. Optional setUp/tearDown methods are overridden from TestCase.
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
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

junit.framework

Class TestCase

```
java.lang.Object
  |
  +--junit.framework.Assert
        |
        +--junit.framework.TestCase
```

All Implemented Interfaces:

Test

Direct Known Subclasses:

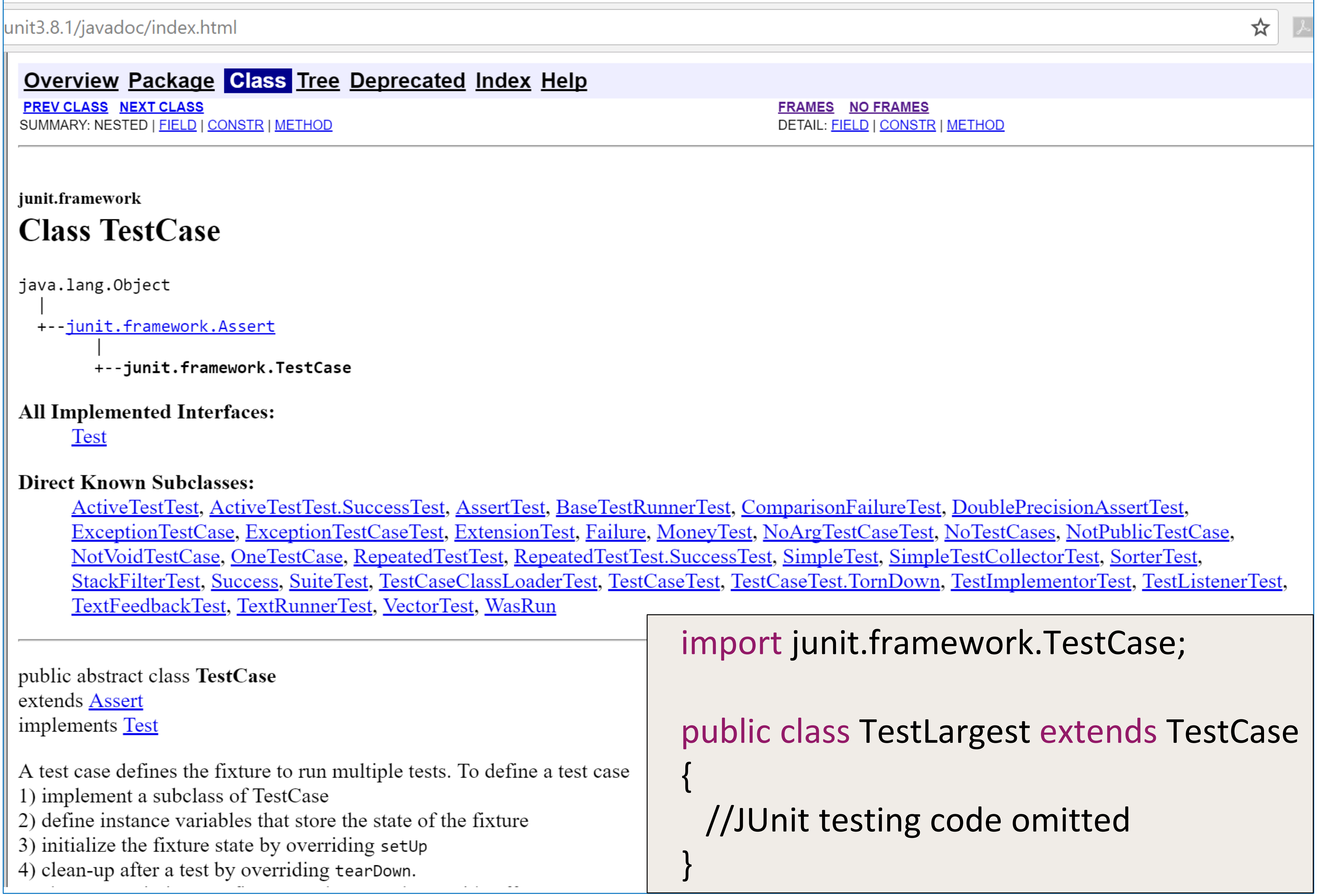
ActiveTestTest, ActiveTestTest.SuccessTest, AssertTest, BaseTestRunnerTest, ComparisonFailureTest, DoublePrecisionAssertTest, ExceptionTestCase, ExceptionTestCaseTest, ExtensionTest, Failure, MoneyTest, NoArgTestCaseTest, NoTestCases, NotPublicTestCase, NotVoidTestCase, OneTestCase, RepeatedTestTest, 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`.

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

1. Test Class must extend TestCase



unit3.8.1/javadoc/index.html
Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

junit.framework

Class TestCase

```
java.lang.Object
  |
  +--junit.framework.Assert
        |
        +--junit.framework.TestCase
```

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

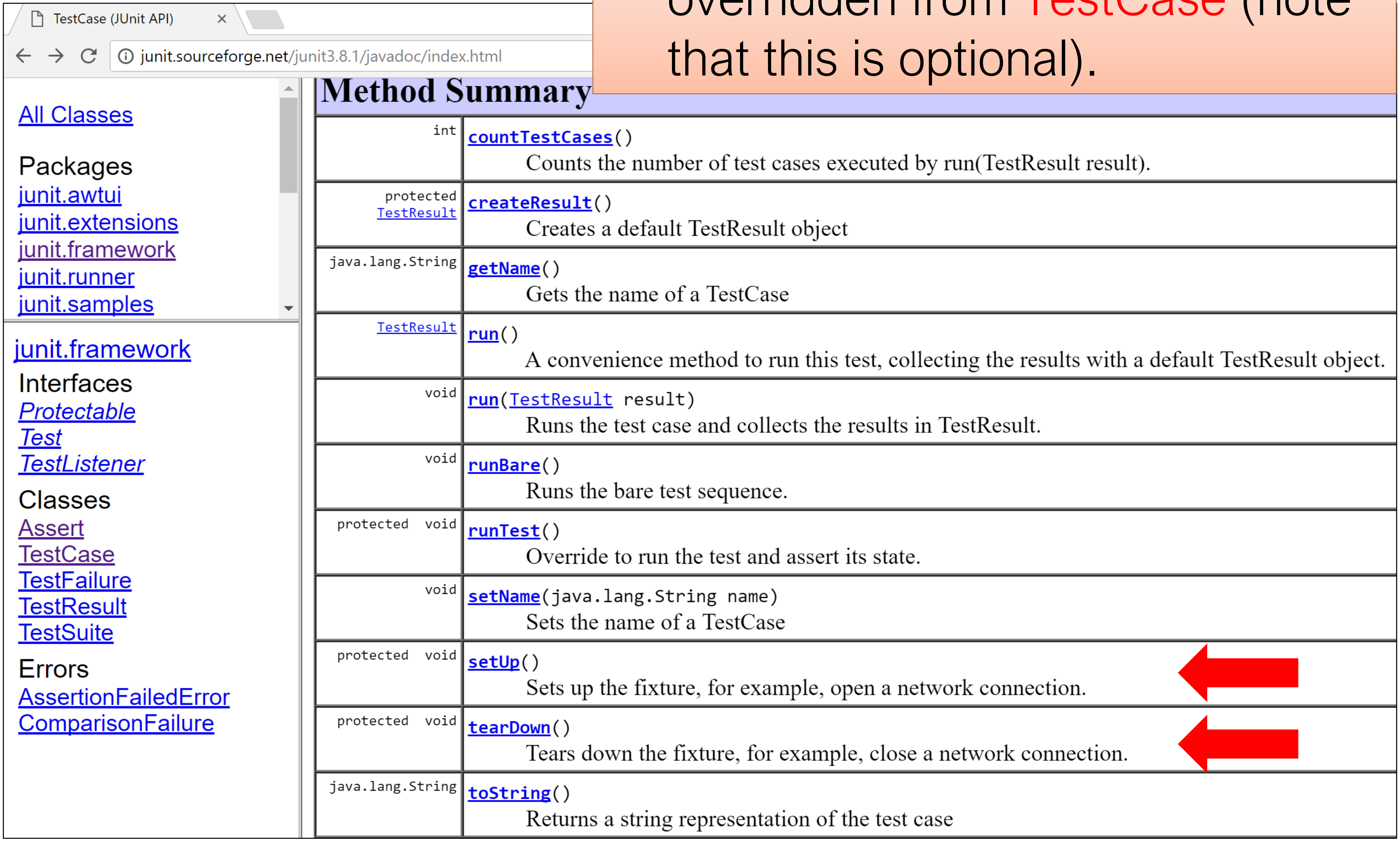
A test case defines the fixture to run multiple tests. To define a test case
1) implement a subclass of TestCase
2) define instance variables that store the state of the fixture
3) initialize the fixture state by overriding `setUp`
4) clean-up after a test by overriding `tearDown`.

```
import junit.framework.TestCase;

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

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

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

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

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.

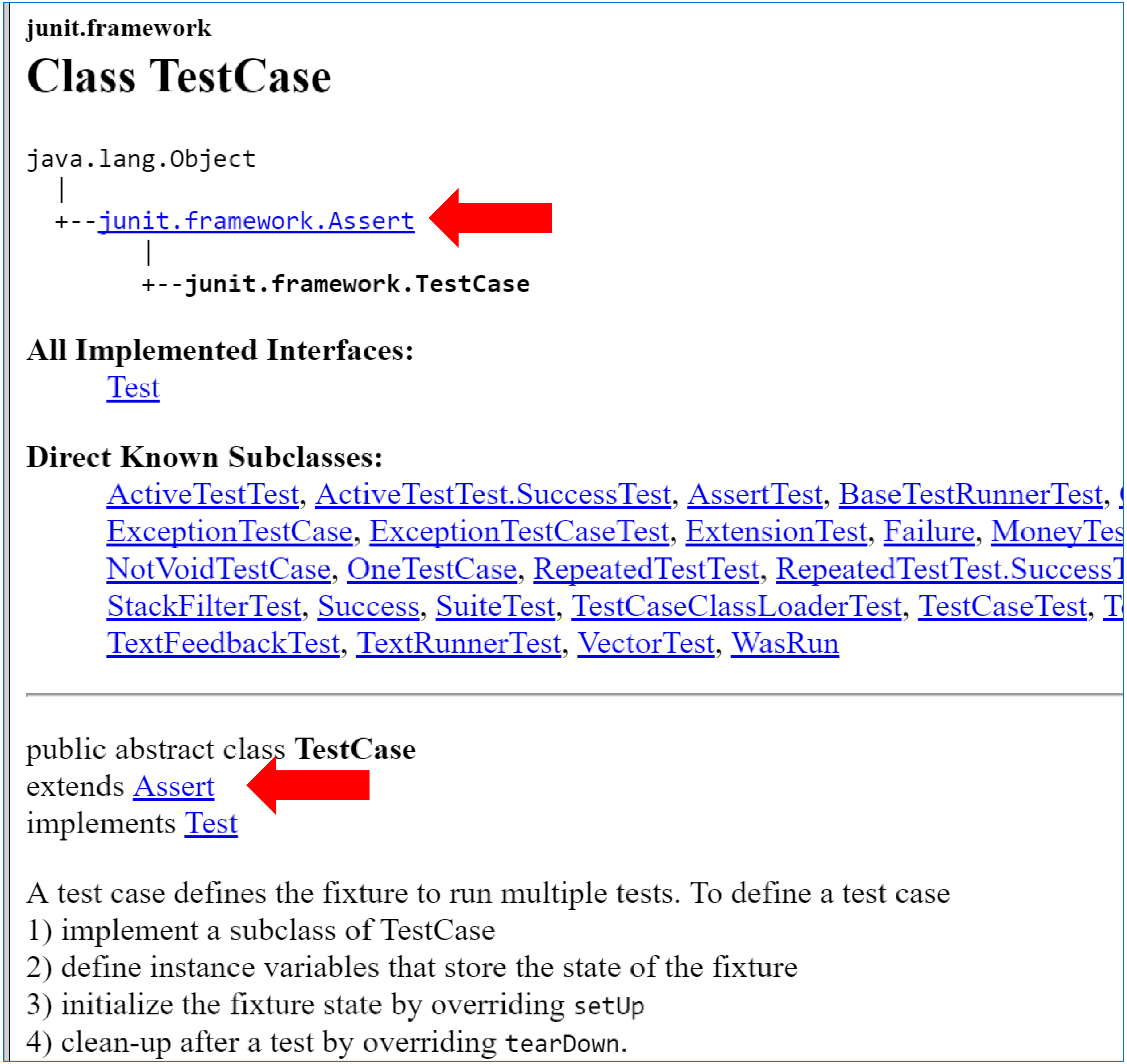
```
import junit.framework.TestCase;

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
  |
  +--junit.framework.Assert
        |
        +--junit.framework.TestCase
```

All Implemented Interfaces:

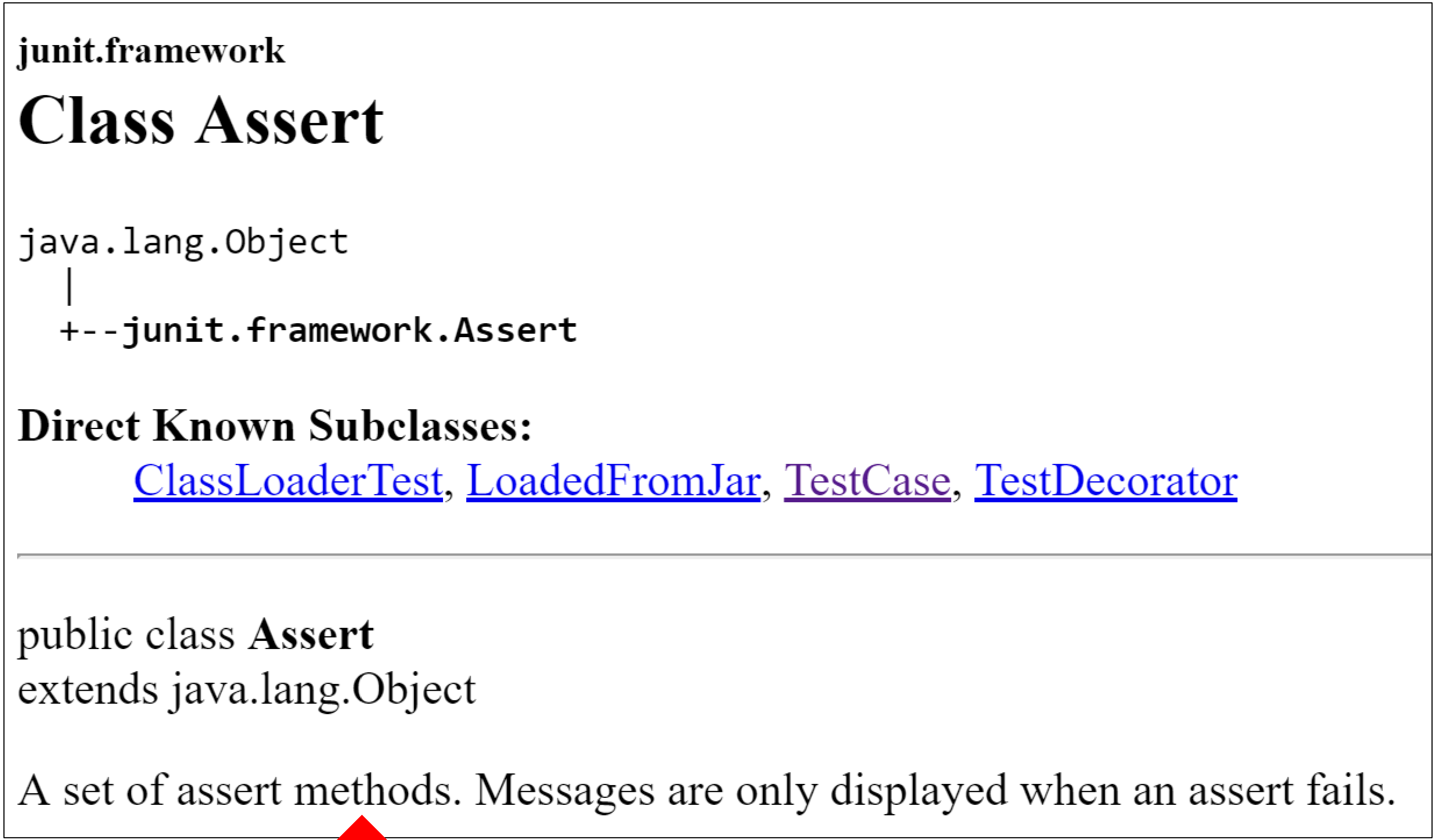
Test

Direct Known Subclasses:

ActiveTestTest, ActiveTestTest.SuccessTest, AssertTest, BaseTestRunnerTest,
ExceptionTestCase, ExceptionTestCaseTest, ExtensionTest, Failure, MoneyTes
NotVoidTestCase, OneTestCase, RepeatedTestTest, RepeatedTestTest.SuccessT
StackFilterTest, Success, SuiteTest, TestCaseClassLoaderTest, TestCaseTest, T
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
```

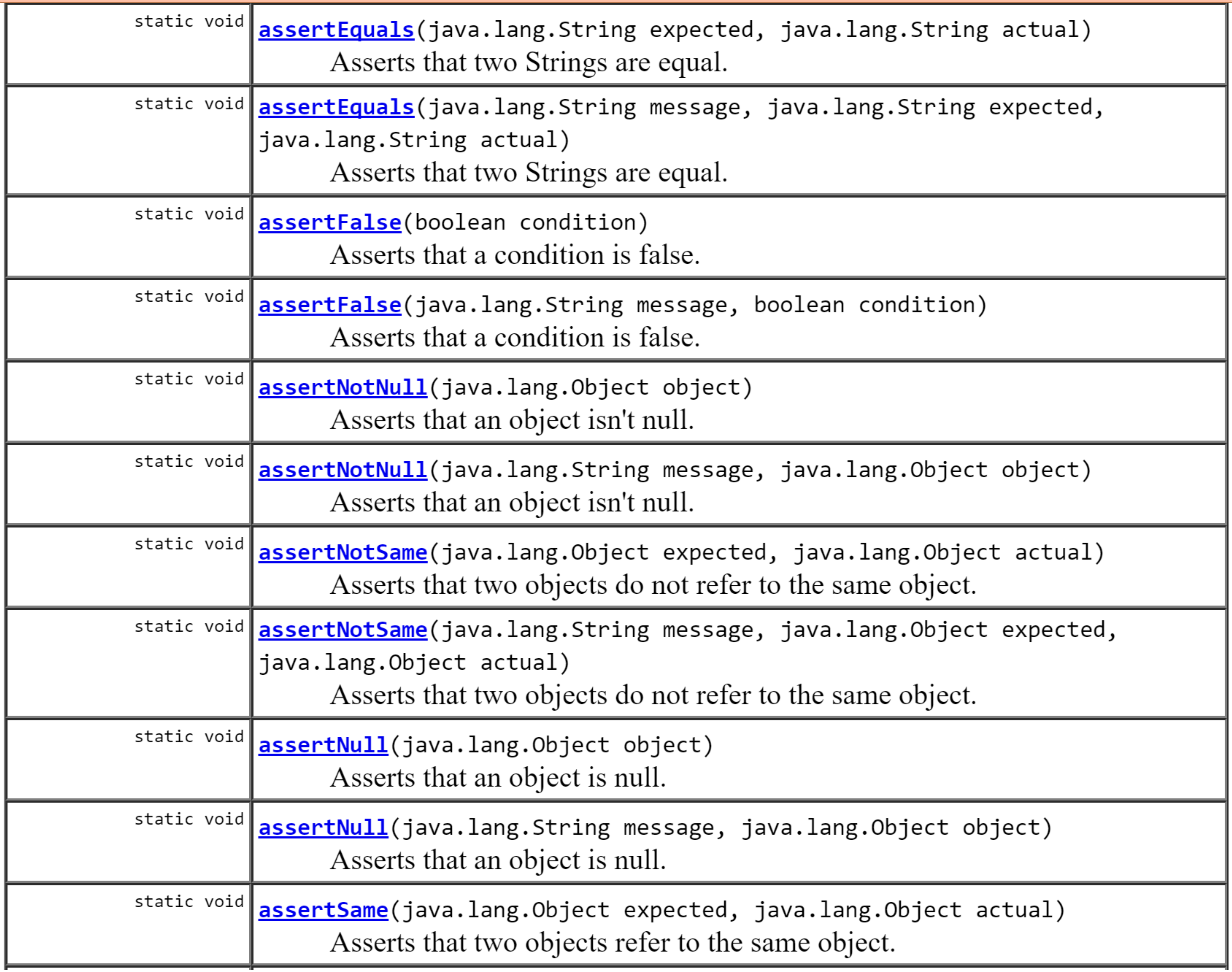
Direct Known Subclasses:

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

JUnit3 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**

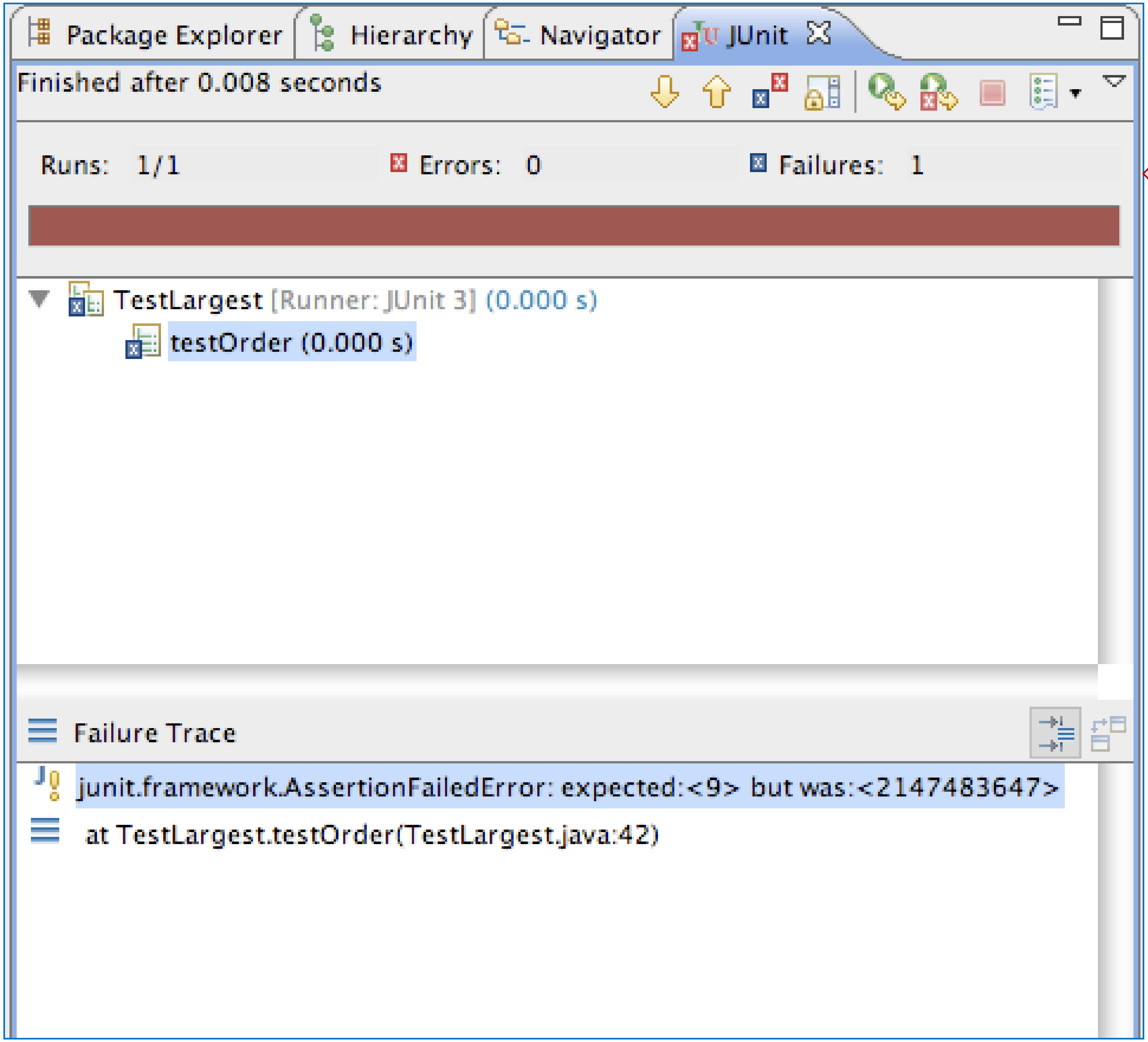
- It has one Unit Test (testOrder) - to verify the behaviour of the largest method.

```
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



Finished after 0.008 seconds

Runs: 1/1 Errors: 0 Failures: 1

TestLargest [Runner: JUnit 3] (0.000 s)

testOrder (0.000 s)

Failure Trace

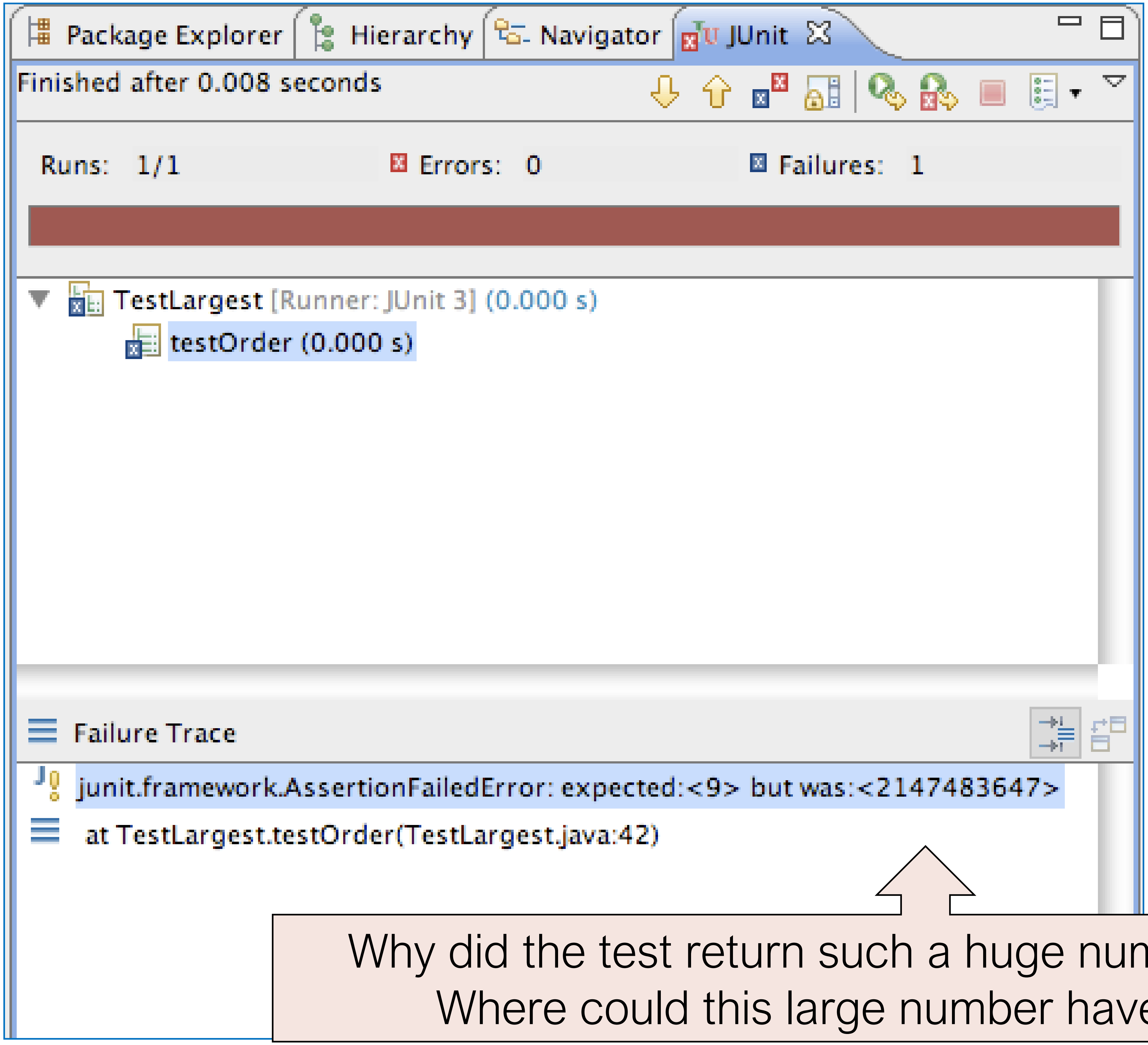
```
junit.framework.AssertionFailedError: expected:<9> but was:<2147483647>
at TestLargest.testOrder(TestLargest.java:42)
```

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



Finished after 0.008 seconds

Runs: 1/1   Errors: 0   Failures: 1

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

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

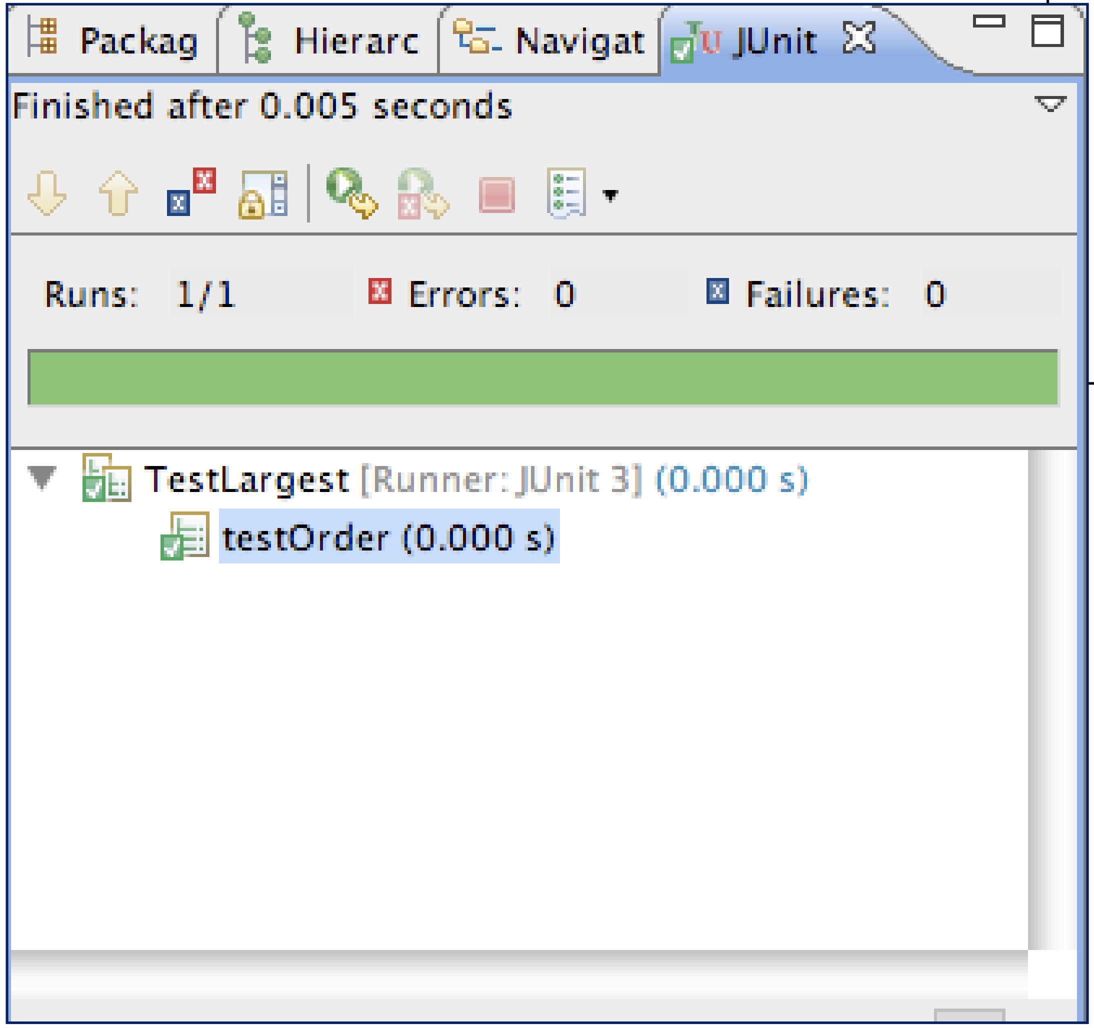
Why did the test return such a huge number instead of 9?
Where could this large number have come from?

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;
}
```



Finished after 0.005 seconds

Runs: 1/1 Errors: 0 Failures: 0

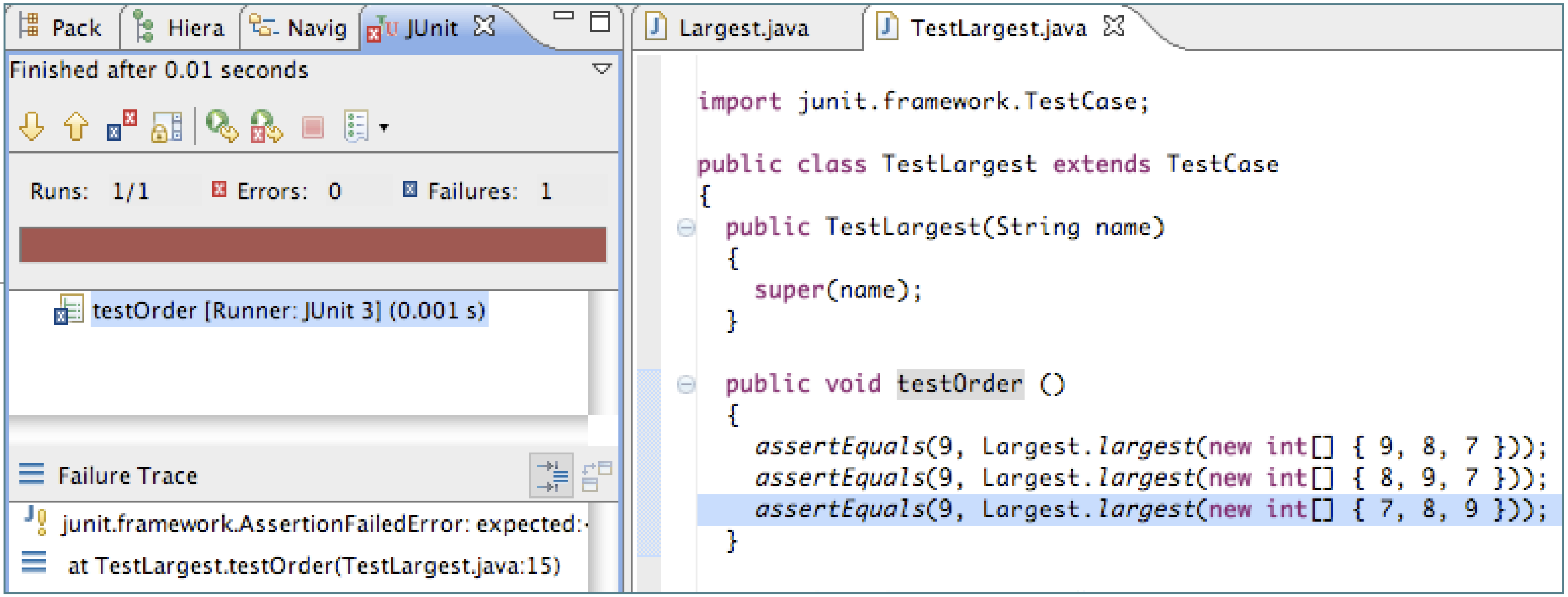
TestLargest [Runner: JUnit 3] (0.000 s)
testOrder (0.000 s)

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

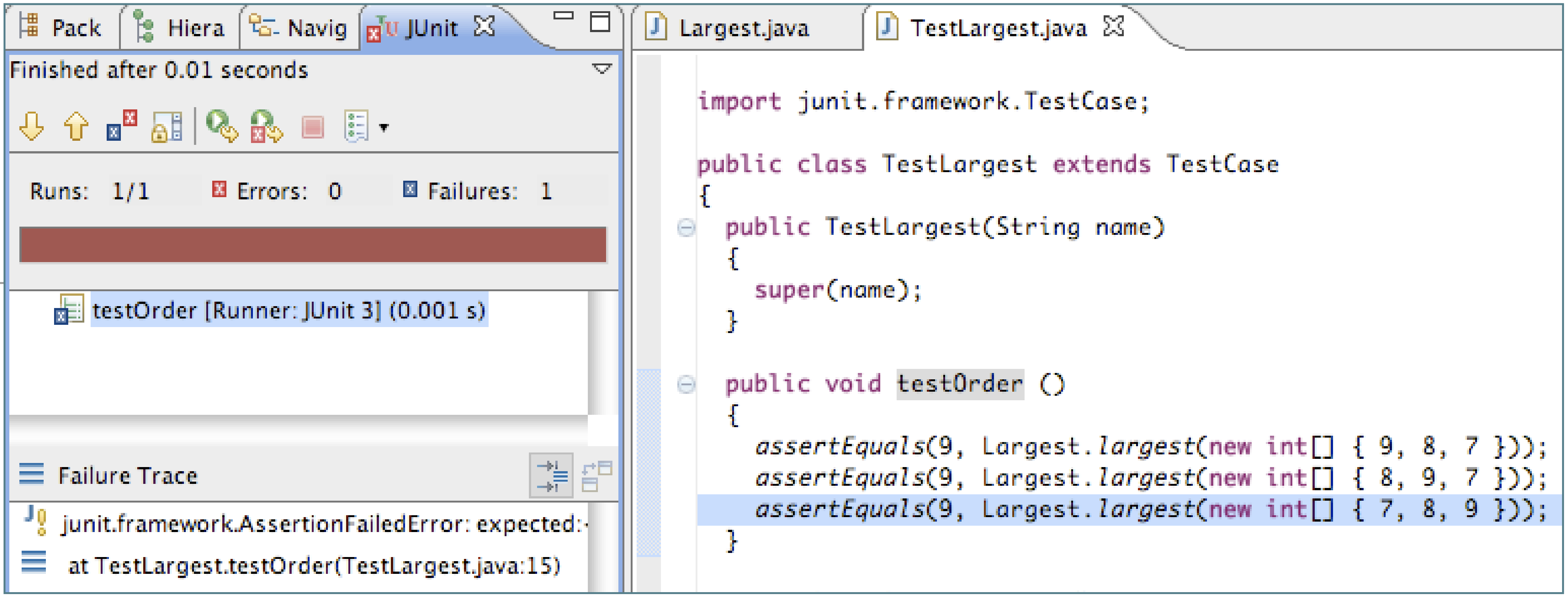
```
junit.framework.AssertionFailedError: expected:
at TestLargest.testOrder(TestLargest.java:15)
```

```
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



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 }));
  }
```

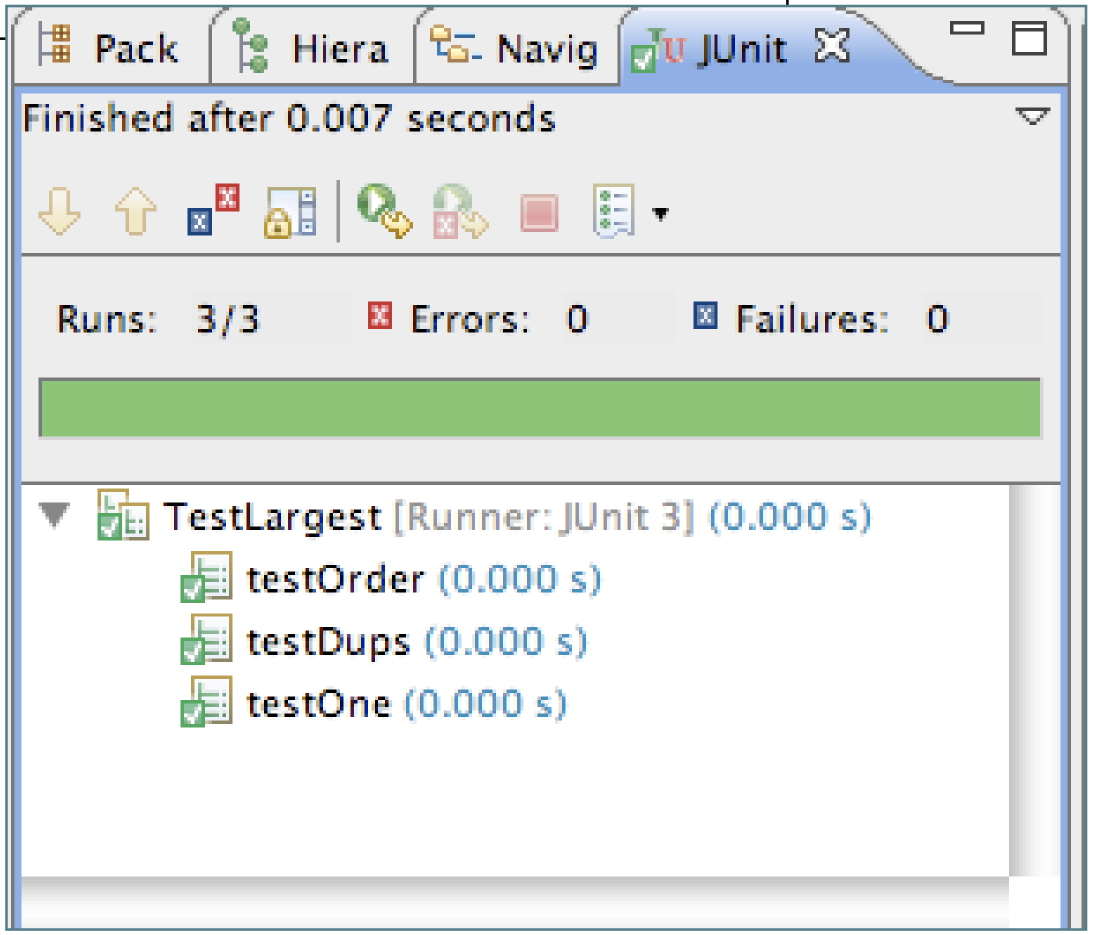
```
public static int largest (int[] list)
{
  int index, max = 0;
  //for (index = 0; index < list.length - 1; index++)
  for (index = 0; index < list.length; index++)
  {
    if (list[index] > max)
    {
      max = list[index];
    }
  }
  return max;
}
```

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



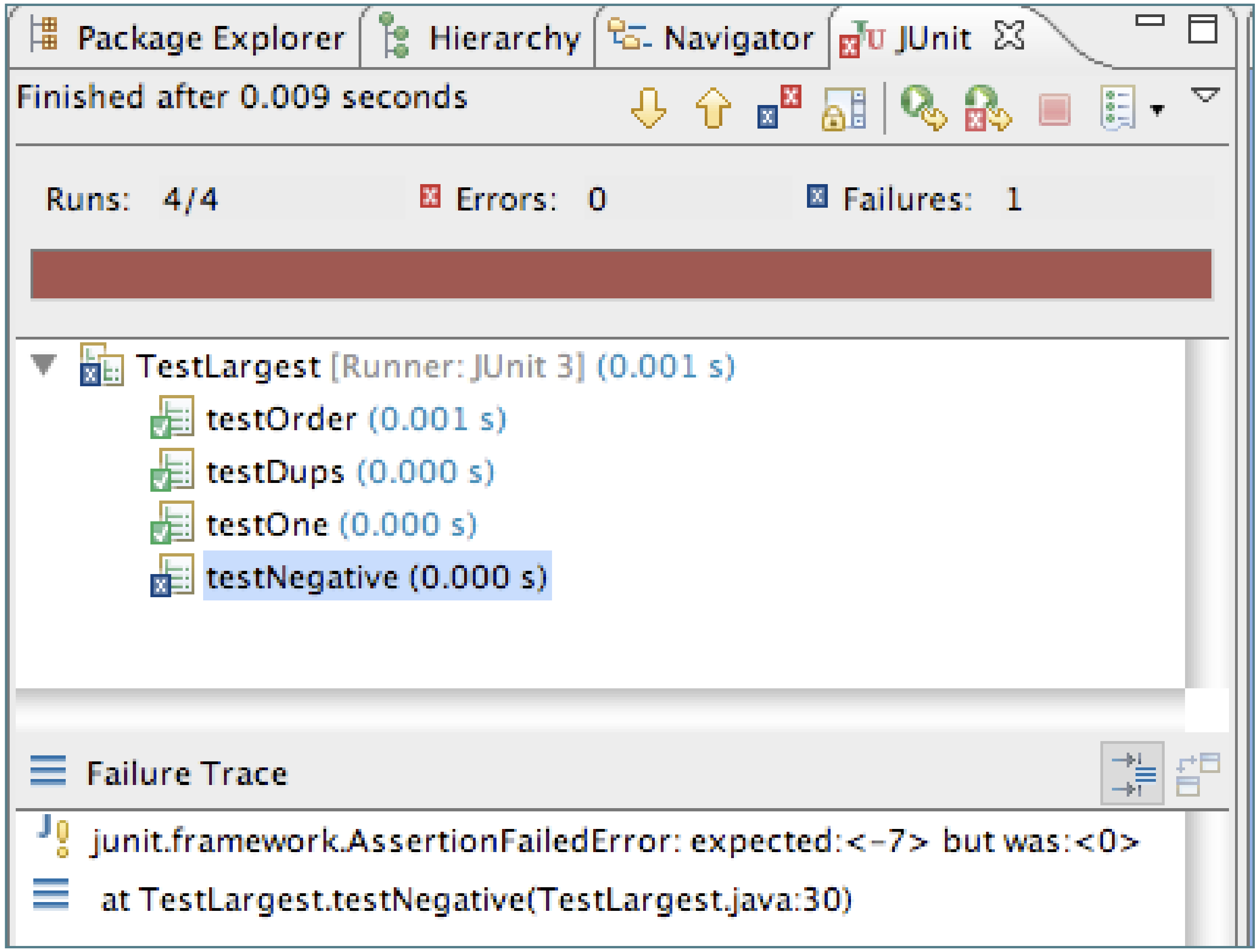
Finished after 0.007 seconds

Runs: 3/3 Errors: 0 Failures: 0

```
TestLargest [Runner: JUnit 3] (0.000 s)
    testOrder (0.000 s)
    testDups (0.000 s)
    testOne (0.000 s)
```

Failure on testNegative

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



```
Finished after 0.009 seconds

Runs:  4/4          Errors:  0          Failures:  1

▼ TestLargest [Runner: JUnit 3] (0.001 s)
      testOrder (0.001 s)
      testDups (0.000 s)
      testOne (0.000 s)
      testNegative (0.000 s)
```

```
junit.framework.AssertionFailedError: expected:<-7> but was:<0>
 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 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;
}
```

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

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

https://www.agilealliance.org/glossary/tdd/

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.

https://www.agilealliance.org/glossary/tdd/

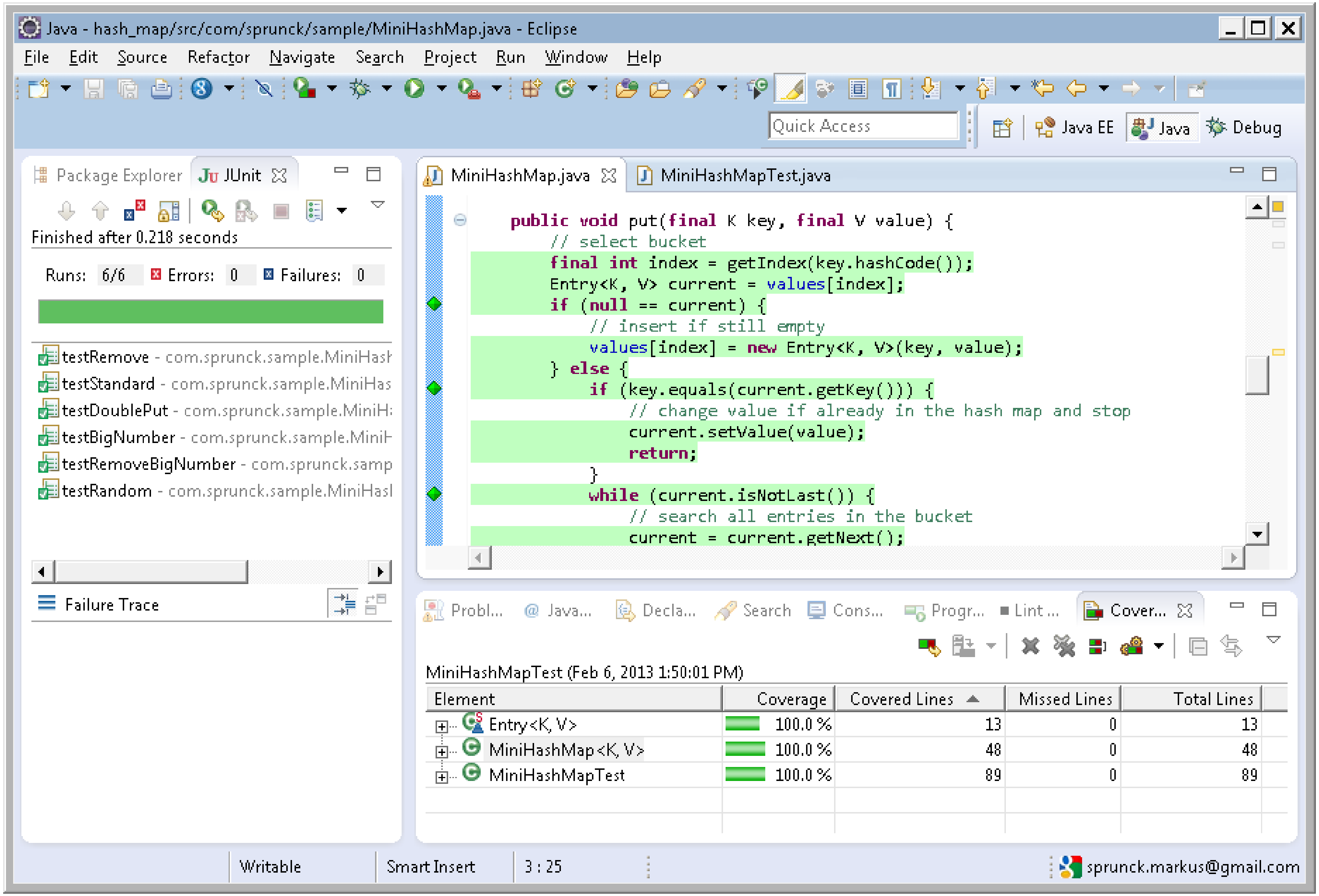
TDD – Signs of Use

- "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.

https://www.agilealliance.org/glossary/tdd/

TDD – Code Coverage – 100% Example



Finished after 0.218 seconds

Runs: 6/6 Errors: 0 Failures: 0

- testRemove - com.sprunck.sample.MiniHash
- testStandard - com.sprunck.sample.MiniHas
- testDoublePut - com.sprunck.sample.MiniH
- testBigNumber - com.sprunck.sample.MiniH
- testRemoveBigNumber - com.sprunck.samp
- testRandom - com.sprunck.sample.MiniHas

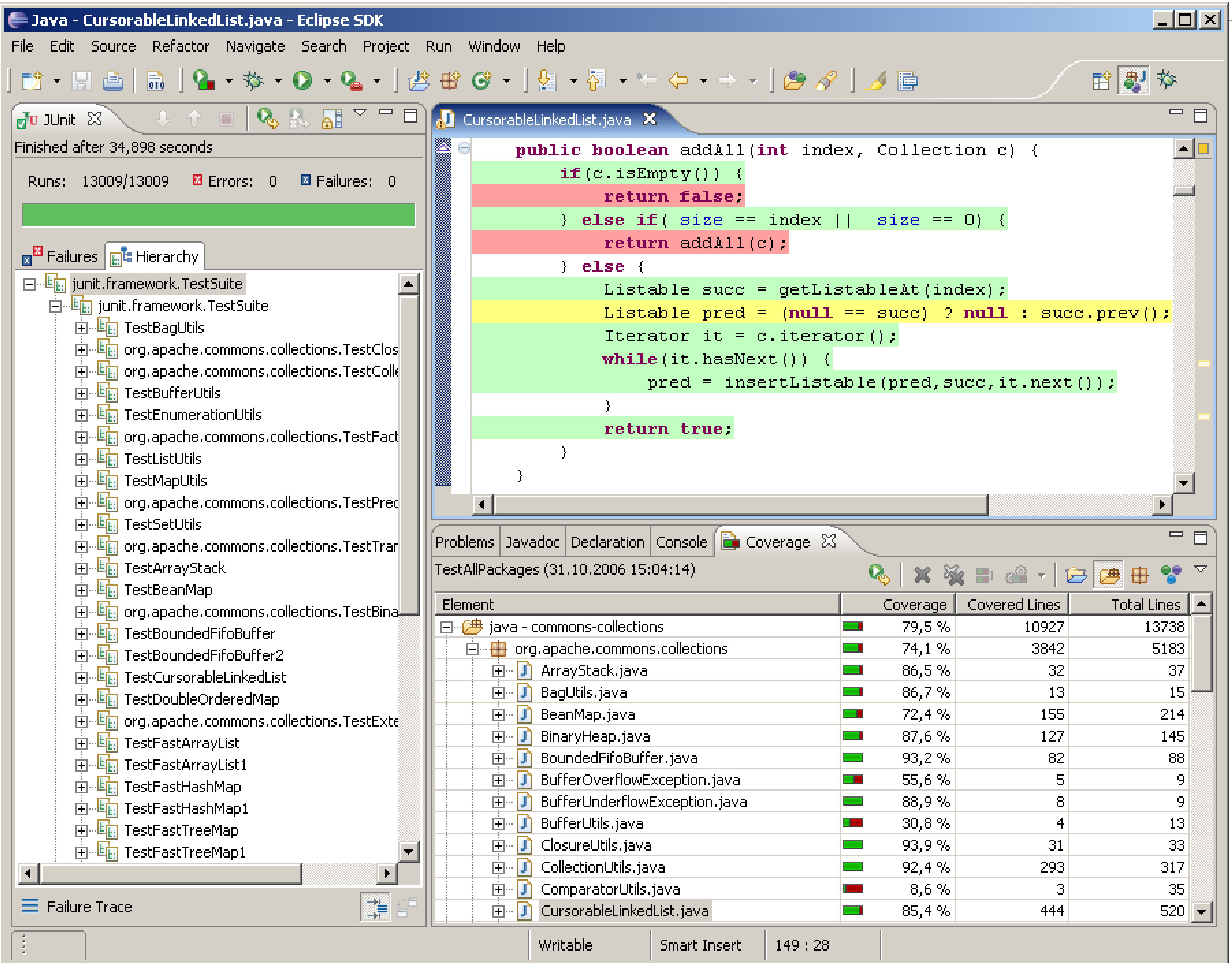
```java
    public void put(final K key, final V value) {
        // select bucket
        final int index = getIndex(key.hashCode());
        Entry<K, V> current = values[index];
        if (null == current) {
            // insert if still empty
            values[index] = new Entry<K, V>(key, value);
        } 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();
```

MiniHashMapTest (Feb 6, 2013 1:50:01 PM)

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

Writable Smart Insert 3 : 25 sprunck.markus@gmail.com

TDD – Code Coverage – 85.4% Example


Java - CursorableLinkedList.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit
CursorableLinkedList.java
Failures
Hierarchy
Failure Trace
Problems Javadoc Declaration Console Coverage
Writable Smart Insert 149 : 28

Finished after 34,898 seconds

Runs: 13009/13009 Errors: 0 Failures: 0

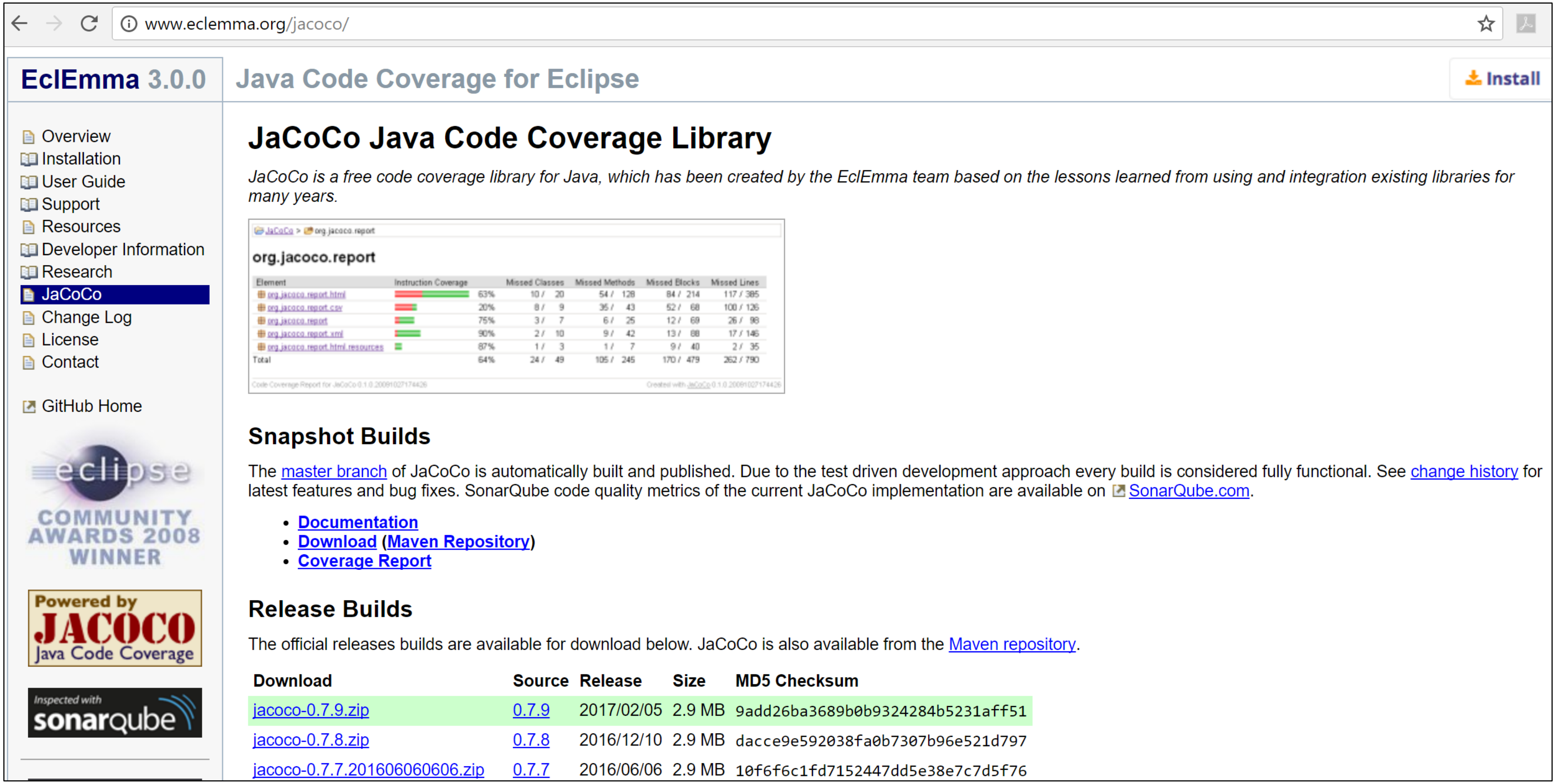
- junit.framework.TestSuite
  - junit.framework.TestSuite
    - TestBagUtils
    - org.apache.commons.collections.TestClos
    - org.apache.commons.collections.TestColle
    - TestBufferUtils
    - TestEnumerationUtils
    - org.apache.commons.collections.TestFact
    - TestListUtils
    - TestMapUtils
    - org.apache.commons.collections.TestPred
    - TestSetUtils
    - org.apache.commons.collections.TestTran
    - TestArrayStack
    - TestBeanMap
    - org.apache.commons.collections.TestBina
    - TestBoundedFifoBuffer
    - TestBoundedFifoBuffer2
    - TestCursorableLinkedList
    - TestDoubleOrderedMap
    - org.apache.commons.collections.TestExte
    - TestFastArrayList
    - TestFastArrayList1
    - TestFastHashMap
    - TestFastHashMap1
    - TestFastTreeMap
    - TestFastTreeMap1

```java
    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;
        }
    }
```

TestAllPackages (31.10.2006 15:04:14)

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/
Install

EclEmma 3.0.0

Java Code Coverage for Eclipse

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

eclipse
COMMUNITY AWARDS 2008 WINNER
Powered by JACOCO Java Code Coverage
Inspected with sonarqube

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.

JaCoCo > org.jacoco.report

org.jacoco.report

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%	8 / 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.

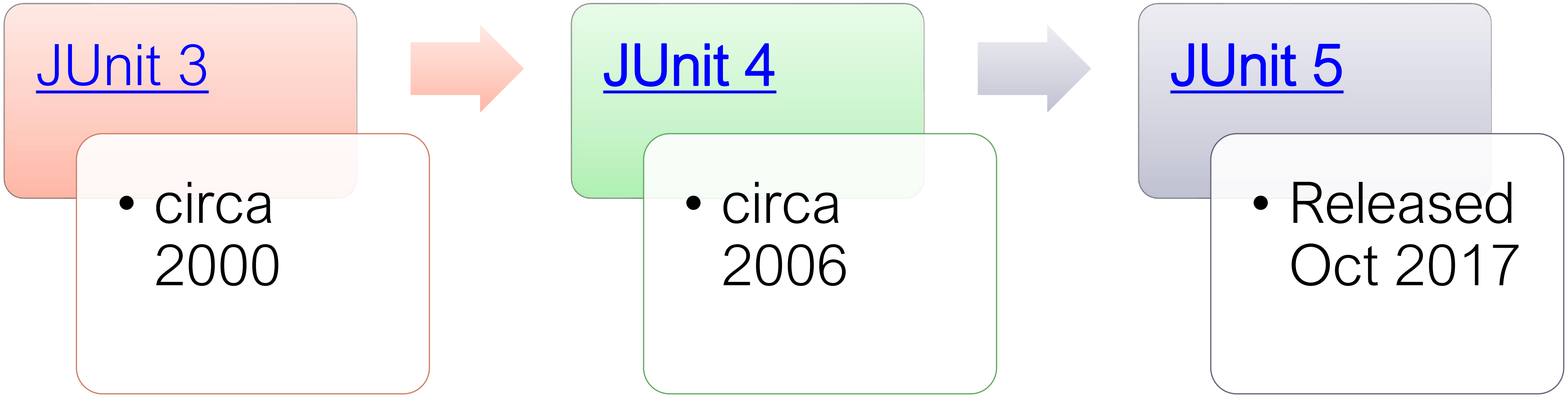
- **Documentation**
- **Download (Maven Repository)**
- **Coverage Report**

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

JUnit Versions



JUnit 3

- circa 2000

JUnit 4

- circa 2006

JUnit 5

- Released Oct 2017

We've looked at JUnit3 in this lecture.
We will add JUnit5 into pacemaker in this week's labs.
We will cover the theory relating to JUnit4 and JUnit5 in future lectures.

http://wiki.c2.com/?TenYearsOfTestDrivenDevelopment

http://junit.sourceforge.net/doc/testinfected/testing.htm

Any questions?