**MOCKITO**

JUnit

**GitHub Repository**

*https://github.com/in28minutes?tab=repositories*

*https://github.com/in28minutes/MockitoTutorialForBeginners*

*https://github.com/in28minutes/JUnitIn28Minutes*

Junit

https://github.com/in28minutes/JUnitIn28Minutes

## Get started with Junit

* [Course Overview](https://github.com/in28minutes/JUnitIn28Minutes" \l "topics)
  + [Section 1 Foundation](https://github.com/in28minutes/JUnitIn28Minutes" \l "section-1-foundation)
  + [Section 2 First JUnit Example](https://github.com/in28minutes/JUnitIn28Minutes" \l "section-2-first-junit-example)
  + [Section 3 Second JUnit Example](https://github.com/in28minutes/JUnitIn28Minutes" \l "section-3-second-junit-example)
  + [Section 4 JUnit Simple Test Scenarios](https://github.com/in28minutes/JUnitIn28Minutes" \l "section-4-junit-simple-test-scenarios)
  + [Section 5 JUnit Intermediate Test Scenarios](https://github.com/in28minutes/JUnitIn28Minutes" \l "section-5-junit-intermediate-test-scenarios)
  + [Section 6 JUnit Best Practices](https://github.com/in28minutes/JUnitIn28Minutes" \l "section-6-junit-best-practices)
  + [Starting Examples](https://github.com/in28minutes/JUnitIn28Minutes" \l "starting-examples)
* [About in28Minutes](https://github.com/in28minutes/JUnitIn28Minutes" \l "about-in28minutes)
  + [Our Beliefs](https://github.com/in28minutes/JUnitIn28Minutes" \l "our-beliefs)
  + [Our Approach](https://github.com/in28minutes/JUnitIn28Minutes" \l "our-approach)
  + [Find Us](https://github.com/in28minutes/JUnitIn28Minutes" \l "useful-links)
  + [Other Courses](https://github.com/in28minutes/JUnitIn28Minutes" \l "other-courses)

## Topics

### Section 1 Foundation

* What is JUnit?
* Why Unit Testing?

### Section 2 First JUnit Example

@Test Annotation

* Running JUnit
* No Failure = Success
* Basic Assert methods

### Section 3 Second JUnit Example

* assertTrue and assertFalse methods
* @Before @After annotations
* @BeforeClass @AfterClass annotations

### Section 4 JUnit Simple Test Scenarios

* Comparing Arrays
* Testing Exceptions
* Performance Unit Tests

### Section 5 JUnit Intermediate Test Scenarios

* Parameterized Tests
* Test Suites

### Section 6 JUnit Best Practices

* Naming Test Methods
* Highlight Important Values in Tests
* Handle Exceptions Properly
* Readable Assert Statements

### Starting Examples

*package com.in28minutes.junit.helper;*

*public class StringHelper {*

*public String truncateAInFirst2Positions(String str) {*

*if (str.length() <= 2)*

*return str.replaceAll("A", "");*

*String first2Chars = str.substring(0, 2);*

*String stringMinusFirst2Chars = str.substring(2);*

*return first2Chars.replaceAll("A", "")*

*+ stringMinusFirst2Chars;*

*}*

*public boolean areFirstAndLastTwoCharactersTheSame(String str) {*

*if (str.length() <= 1)*

*return false;*

*if (str.length() == 2)*

*return true;*

*String first2Chars = str.substring(0, 2);*

*String last2Chars = str.substring(str.length() - 2);*

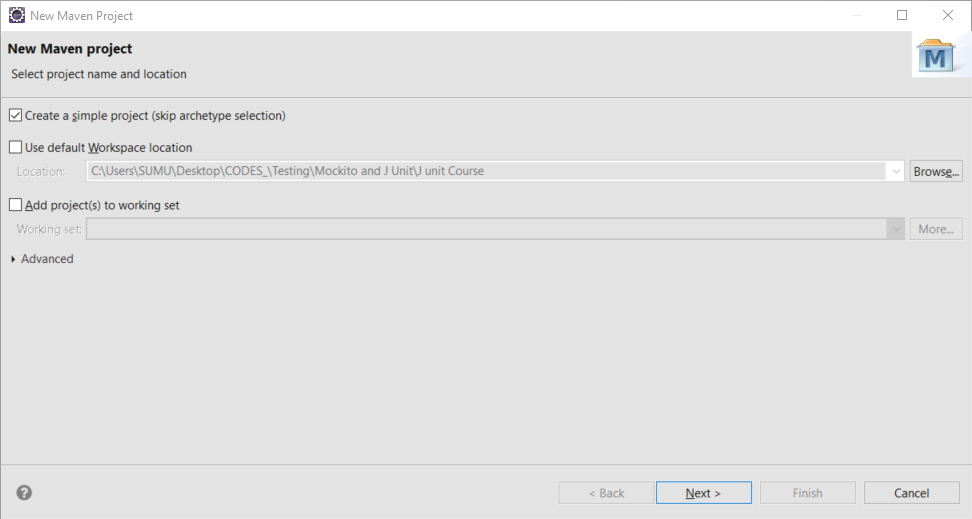
*return first2Chars.equals(last2Chars);*

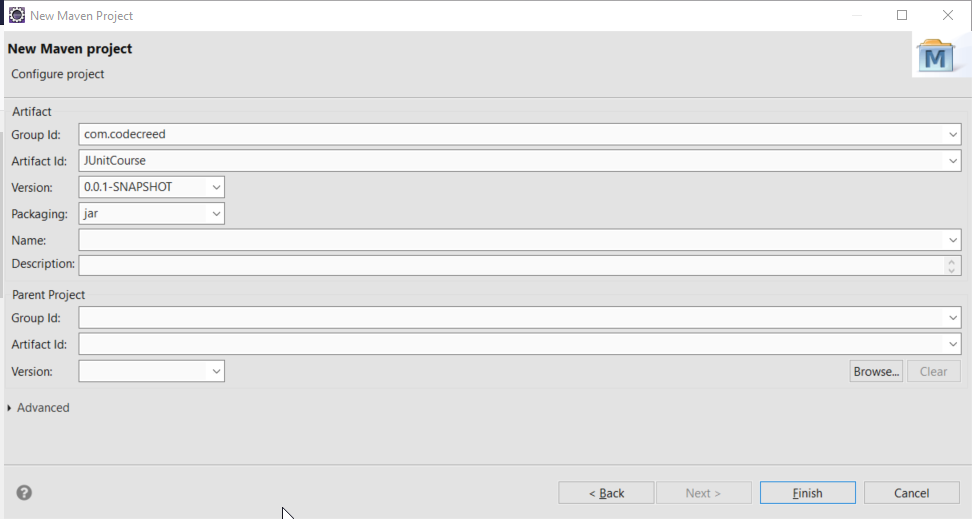
*}*

*}*

The advantage of unit testing is that I can specially check for a part of code (ie methods) within milliseconds.

Writing Great Unit Tests distinguishes Good Programmers from Great Programmers.





Starter example code

https://github.com/in28minutes/JUnitIn28Minutes

package com.in28minutes.junit.helper;

public class StringHelper {

public String truncateAInFirst2Positions(String str) {

if (str.length() <= 2)

return str.replaceAll("A", "");

String first2Chars = str.substring(0, 2);

String stringMinusFirst2Chars = str.substring(2);

return first2Chars.replaceAll("A", "")

+ stringMinusFirst2Chars;

}

public boolean areFirstAndLastTwoCharactersTheSame(String str) {

if (str.length() <= 1)

return false;

if (str.length() == 2)

return true;

String first2Chars = str.substring(0, 2);

String last2Chars = str.substring(str.length() - 2);

return first2Chars.equals(last2Chars);

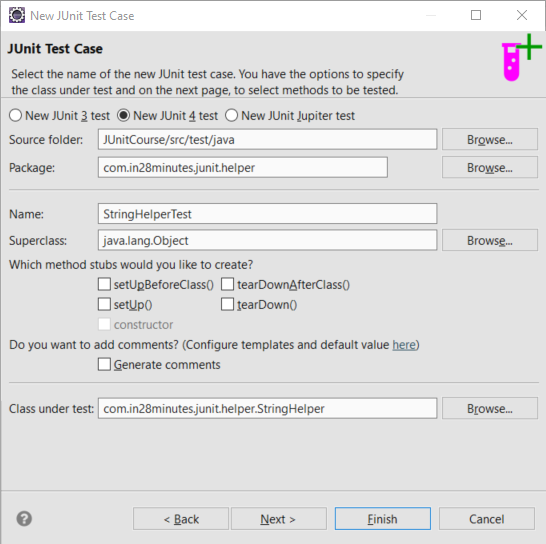
}

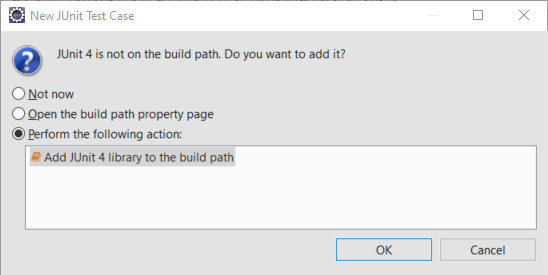
}

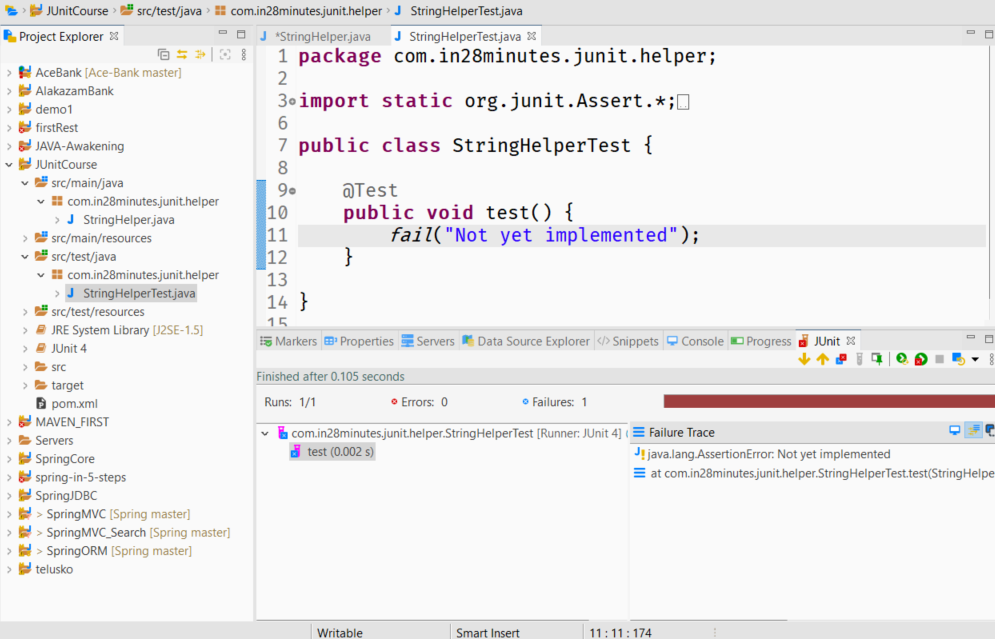
paste in src/main/java

and automagically alll the packages and classes will be creates

create a new J Unit Test class in the /src/test/java



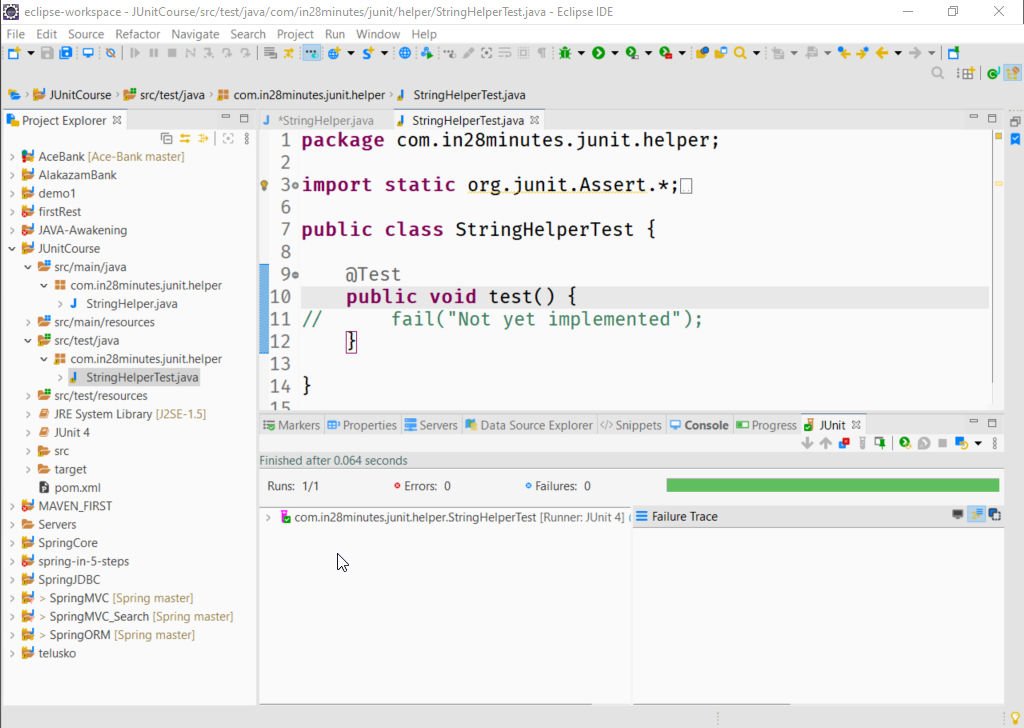




### fail()

run as > junit test case

red bar becaus of **fail()** method



if there are no failures or expectation then a test is said to be a success.

So if we don’t test anything the test will always pass and that's not a good idea.

So if we comment the fail() call the test is success {the **green** bar}

fail|() is one of the method in a series of test methods and it can fail a test.

### assertEquals()

To check two values are equal or not.

package com.in28minutes.junit.helper;

import static org.junit.Assert.\*;

import org.junit.Test;

public class StringHelperTest {

@Test

public void test() {

// fail("Not yet implemented");

assertEquals("ABC", "ABCD");

}

}

assertEquals("ABC", "ABCD");

//**expected, actual**

org.junit.ComparisonFailure: expected:<ABC[]> but was:<ABC[D]>

at org.junit.Assert.assertEquals(Assert.java:117)

....

assertEquals("ABCD", "ABCD");

package com.in28minutes.junit.helper;

import static org.junit.Assert.\*;

import org.junit.Test;

public class StringHelperTest {

@Test

public void test() {

// fail("Not yet implemented");

// assertEquals("ABCD", "ABCD");

StringHelper helper = new StringHelper();

String actual = helper.truncateAInFirst2Positions("AACD");

String expected = "CD";

assertEquals(**expected** ,**actual**);

}

}

//refactoring the variable to inline

package com.in28minutes.junit.helper;

import static org.junit.Assert.\*;

import org.junit.Test;

public class StringHelperTest {

@Test

public void test() {

// fail("Not yet implemented");

// assertEquals("ABCD", "ABCD");

StringHelper helper = new StringHelper();

assertEquals(helper.truncateAInFirst2Positions("AACD"), "CD");

}

}

EA

its never a good idea to test 2 or more conditions in the same @Test method

runs => no of test cases

failures => no of test which didn't ran from

when case of multiple @Test methods if we choose to run only a single 2test method -> highlight the method and run as Junit test case.

We can also put the helper object at class level instead of method level.

**package** com.in28minutes.junit.helper;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**public** **class** StringHelperTest {

StringHelper helper = **new** StringHelper();

@Test

**public** **void** testtruncateAInFirst2Positions\_ConditionsOne() {

// fail("Not yet implemented");

// assertEquals("ABCD", "ABCD");

// StringHelper helper = new StringHelper();

// assertEquals("CD",helper.truncateAInFirst2Positions("AACD"));

// its not a good idea to perform to add 2 or more checks in the same unit test

*assertEquals*( "CD",helper.truncateAInFirst2Positions("AACD"));

}

@Test

**public** **void** testtruncateAInFirst2Positions\_ConditionsTwo() {

// public void -> must

// it should be visible to other frameworks which may call

// StringHelper helper = new StringHelper();

*assertEquals*("CD",helper.truncateAInFirst2Positions("AACD"));

}

}



### **assertFalse()**

Checks for false, test passes if the output is false

### **assertTrue()**

test passes if the output is true

\*both the above methods can be overloaded with a String parameter in the beginning

**package** com.in28minutes.junit.helper;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*assertFalse*;

**import** **static** org.junit.Assert.*assertTrue*;

**import** org.junit.Test;

**public** **class** StringHelperTest {

StringHelper helper = **new** StringHelper();

// AACD => CD, ACD => CD ,CDEF=> CDAA=> CDAA

@Test

**public** **void** testTruncateAInFirst2Positions\_ConditionsOne() {

// fail("Not yet implemented");

// assertEquals("ABCD", "ABCD");

// StringHelper helper = new StringHelper();

// assertEquals("CD",helper.truncateAInFirst2Positions("AACD"));

// its not a good idea to perform to add 2 or more checks in the same unit test

*assertEquals*( "CD",helper.truncateAInFirst2Positions("AACD"));

}

@Test

**public** **void** testTruncateAInFirst2Positions\_ConditionsTwo() {

// public void -> must

// public because, it should be visible to other frameworks which may call

// StringHelper helper = new StringHelper();

*assertEquals*("CD",helper.truncateAInFirst2Positions("AACD"));

}

// ABCD => false ; ABAB => true ; AB => true ; A = false

@Test

**public** **void** testAreFirstAndLastTwoCharactersTheSame\_BasicNegativeScenario() {

// assertEquals(false, helper.areFirstAndLastTwoCharactersTheSame("ABC"));

*assertFalse*("MSG: failed for 'false'",helper.areFirstAndLastTwoCharactersTheSame("AB"));

}

@Test

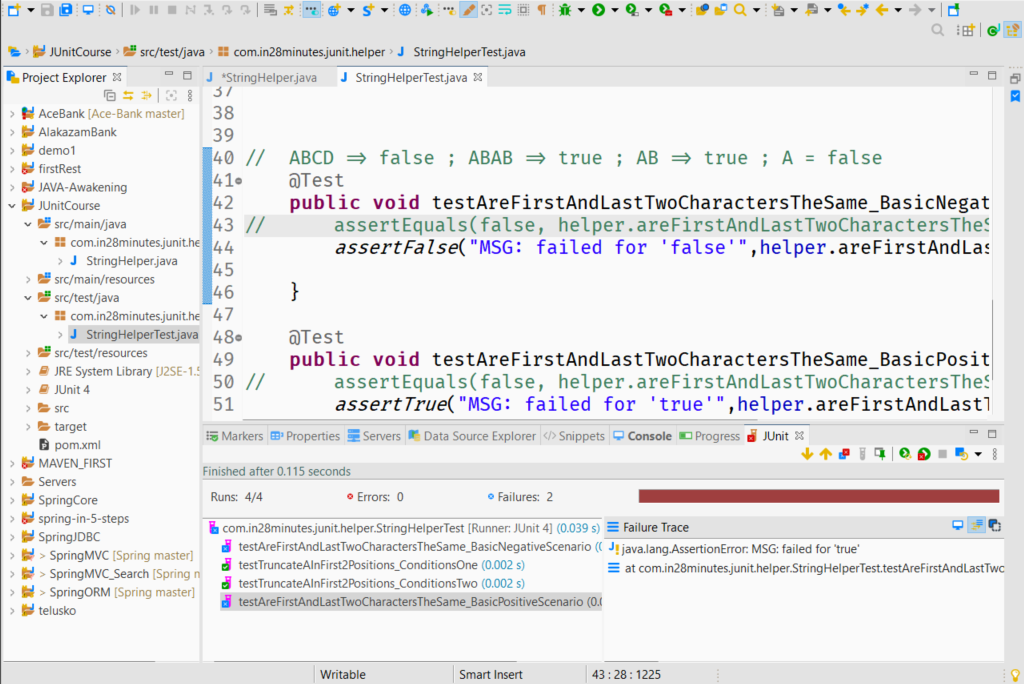
**public** **void** testAreFirstAndLastTwoCharactersTheSame\_BasicPositiveScenario() {

// assertEquals(false, helper.areFirstAndLastTwoCharactersTheSame("ABC"));

*assertTrue*("MSG: failed for 'true'",helper.areFirstAndLastTwoCharactersTheSame("ABC"));

}

}

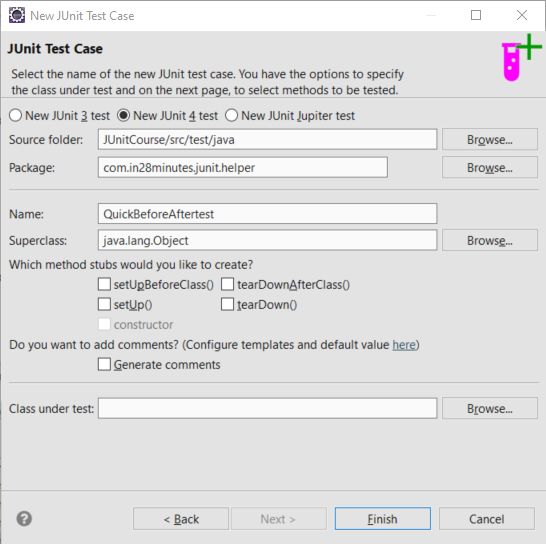


### @Before

This can be used for a method which should be precursor for all the @tests

### @**After**

This can be used for a method which should be successor for all the @tests



**package** com.in28minutes.junit.helper;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

**public** **class** QuickBeforeAftertest {

@Before

**public** **void** setup() {

System.***out***.println("Before test \"Initializers & Precurssors\"");

}

@After

**public** **void** teardown() {

System.***out***.println("After test \"Dispossal\"");

}

@Test

**public** **void** test1() {

System.***out***.println("test 1 excuted");

}

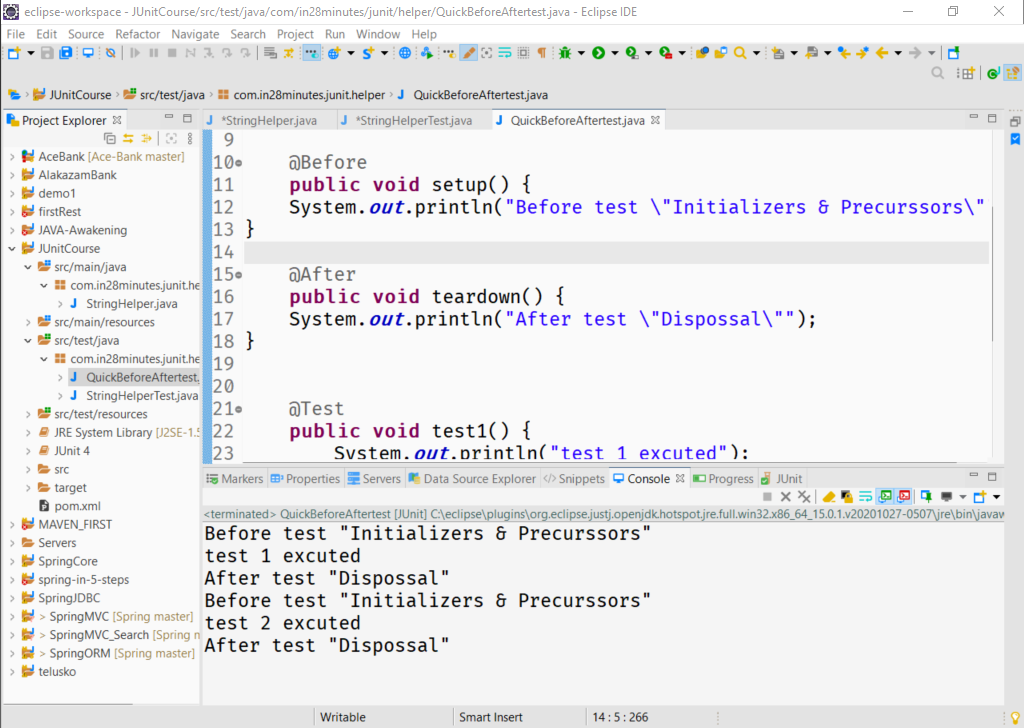
@Test

**public** **void** test2() {

System.***out***.println("test 2 excuted");

}

}



**import** **static** org.junit.Assert.*assertTrue*;

**import** org.junit.Before;

**import** org.junit.Test;

**public** **class** StringHelperTest {

StringHelper helper;

@Before

**public** **void** before() {

helper = **new** StringHelper();

}

// AACD => CD, ACD => CD ,CDEF=> CDAA=> CDAA

@Test

### @**BeforeClass**

This can be used for a method which should be precursor for the @tests

and it executes once only.

The methods should be **public** **static** **void**

### @**AfterClass**

This can be used for a method which should be precursor for the @tests

and it executes once only.

The methods should be **public** **static** **void**

**package** com.in28minutes.junit.helper;

**import** org.junit.After;

**import** org.junit.AfterClass;

**import** org.junit.Before;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**public** **class** QuickBeforeAftertest {

@Before

**public** **void** initialize() {

System.***out***.println("Before test \"Initializers & Precurssors\"");

}

@After

**public** **void** teardown() {

System.***out***.println("After test \"Dispossal\"");

}

@BeforeClass

**public** **static** **void** setup() {

System.***out***.println("\*\*\*Before execution of class\"Initializers & Precurssors\"");

}

@AfterClass

**public** **static** **void** dispose() {

System.***out***.println("\*\*\*After execution of class \"Dispossal\"");

}

@Test

**public** **void** test1() {

System.***out***.println("test 1 excuted");

}

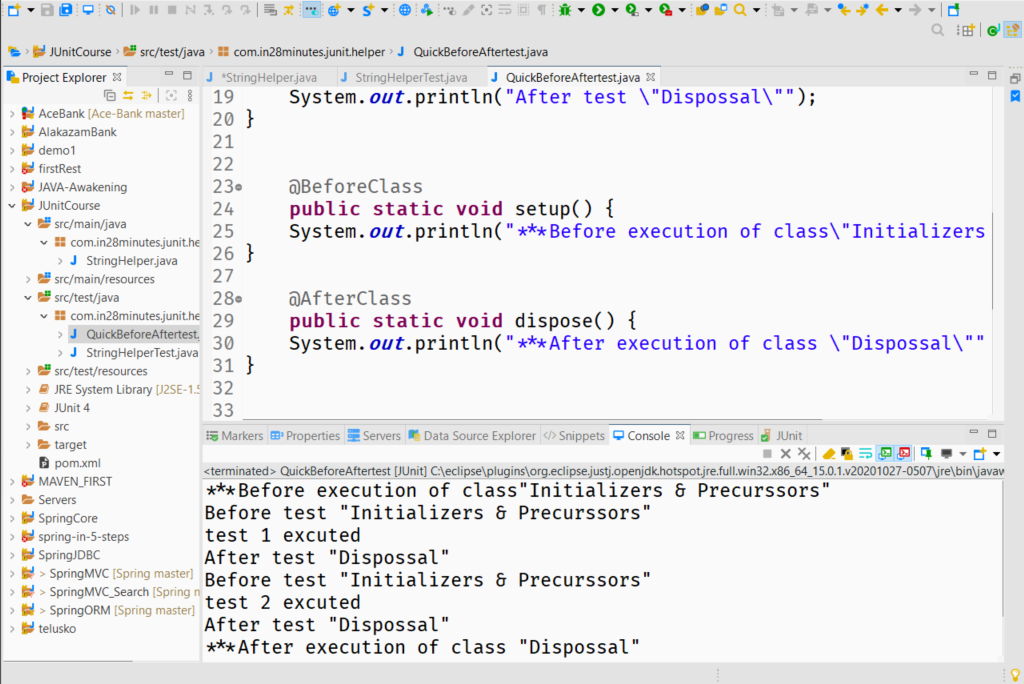
@Test

**public** **void** test2() {

System.***out***.println("test 2 excuted");

}

}

****

### assertEquals(expected,sample)

Checks their reference

### assertArrayEquals(expected,sample)

Checks their values

if negative result, it will also provide like additional details of trace stack.

\*\*\* We can also get Exceptions in the Test Cases,

So we can handle it by try-catch

Or we can mention at @Test annotation, using expected attribute

@Test(expected=NullPointerException.**class**)

but if there is no exception its a failure

also a no failure == success

**package** com.in28minutes.junit.helper;

**import** **static** org.junit.Assert.*assertArrayEquals*;

**import** java.util.Arrays;

**import** org.junit.Test;

**public** **class** ArraysCompareTestJava {

@Test

**public** **void** testArray\_test1() {

**int** sample[]= {15,52,31,41,15,16,177};

**int** expected[]= {15,15,16,31,41,52,177};

Arrays.*sort*(sample);

// assertEquals(expected,sample);// checks the reference

*assertArrayEquals*(expected,sample);// checks the reference

}

@Test

**public** **void** testArray\_test2() {

**int** sample[]= {15,52,31,41,15,16,177};

**int** expected[]= {15,15,16,41,31,52,177};//wrong

/\* arrays first differed at element [3]; expected:<41> but was:<31> at org.junit.internal.ComparisonCriteria.arrayEquals(ComparisonCriteria.java:78)

\*/

Arrays.*sort*(sample);

*assertArrayEquals*(expected,sample);// checks the reference

}

@Test(expected=NullPointerException.**class**)

//if the mentioned exception will come then test pass

**public** **void** testArray\_test3() {

**int** sample[]= **null**;

**int** expected[]= {15,15,16,41,31,52,177};//wrong

Arrays.*sort*(sample);//NullPointerException

*assertArrayEquals*(expected,sample);// checks the reference

}

}

### Performance Testing

we can mention at @Test annotation, using timeout attribute.

package com.in28minutes.junit.helper;

import static org.junit.Assert.assertArrayEquals;

import java.util.Arrays;

import java.util.Iterator;

import org.junit.Test;

public class ArraysCompareTestJava {

@Test

public void testArray\_test1() {

int sample[]= {15,52,31,41,15,16,177};

int expected[]= {15,15,16,31,41,52,177};

Arrays.sort(sample);

// assertEquals(expected,sample);// checks the reference

assertArrayEquals(expected,sample);// checks the reference

}

@Test

public void testArray\_test2() {

int sample[]= {15,52,31,41,15,16,177};

int expected[]= {15,15,16,31,41,52,177};//wrong

Arrays.sort(sample);

assertArrayEquals(expected,sample);// checks the reference

}

@Test(expected=NullPointerException.class)

//if the mentioned exception will come then test pass

public void testArray\_test3() {

int sample[]= null;

int expected[]= {15,15,16,41,31,52,177};//wrong

Arrays.sort(sample);//NullPointerException

// assertArrayEquals(expected,sample);// checks the reference

}

@Test(timeout=1000)//miliseconds

//if the mentioned time limit will surpass , test will fail

public void testSort\_Performance() {

int sample[]= {15,15,16,41,31,52,177};//wrong

for (int i = 0; i < 1000000; i++) {

sample[0]=i;

Arrays.sort(sample);

}

}

}

### Parametrized Test

We can automate things to some extent by Parametrizing the test, Here we will use String arrays en capsule in a collection and using

1. on class

@RunWith(Parameterized.**class**)

**public** **class** StringHelperParameterized {

1. on methods

@Parameters

**public** **static** Collection<String[]> testCondition() {

1. and use of field constructors
   * generate using right click, constructors using fields
   * we can also use ctrl+C , ctrl+v on the classes in the packages.

**package** com.in28minutes.junit.helper;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*assertFalse*;

**import** **static** org.junit.Assert.*assertTrue*;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

@RunWith(Parameterized.**class**)

**public** **class** StringHelperParameterized {

**static** StringHelper *helper*;

**private** String input;

**private** String expected;

**public** StringHelperParameterized(String input, String expected) {

**super**();

**this**.input = input;

**this**.expected = expected;

}

@BeforeClass

**public** **static** **void** before() {

*helper* = **new** StringHelper();

}

@Parameters

**public** **static** Collection<String[]> testCondition() {

String expectedOutputs [][]={{"AACD" ,"CD"},

{"ACD","CD"},{"CDEF","CDEF"},{"CDAA","CDAA"}};

**return** Arrays.*asList*(expectedOutputs);

/\*

\* all the String arrays contents will be passes to constructor and a new test

\* instance will be created

\*/

}

// AACD => CD, ACD => CD ,CDEF=> CDEF ,CDAA=> CDAA

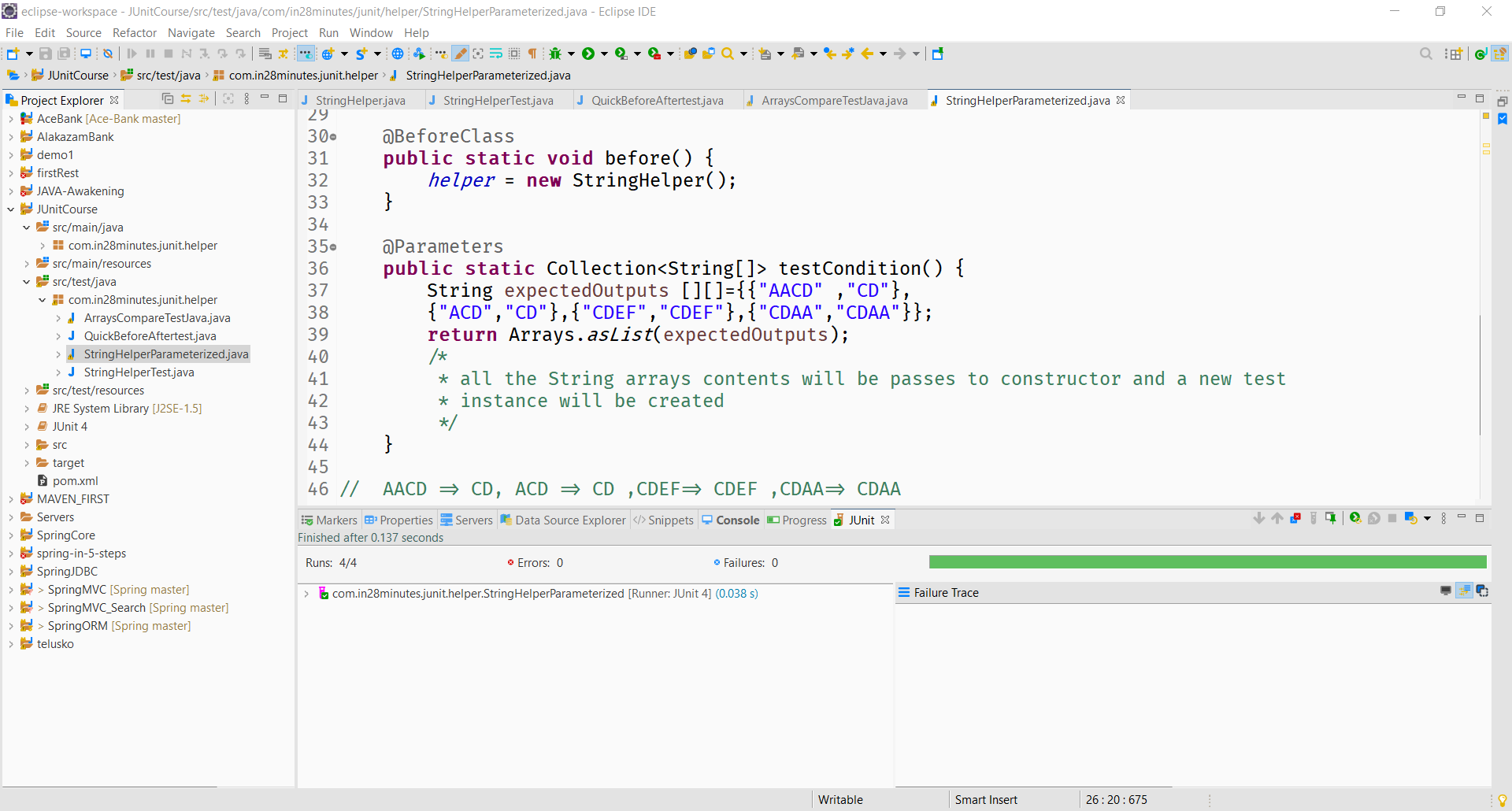
@Test

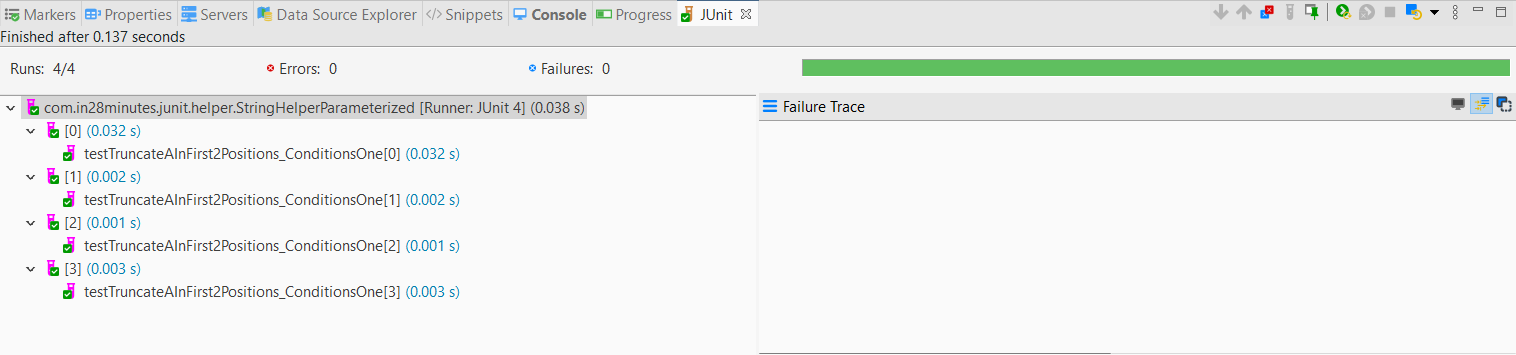
**public** **void** testTruncateAInFirst2Positions\_ConditionsOne() {

*assertEquals*( expected,*helper*.truncateAInFirst2Positions(input));

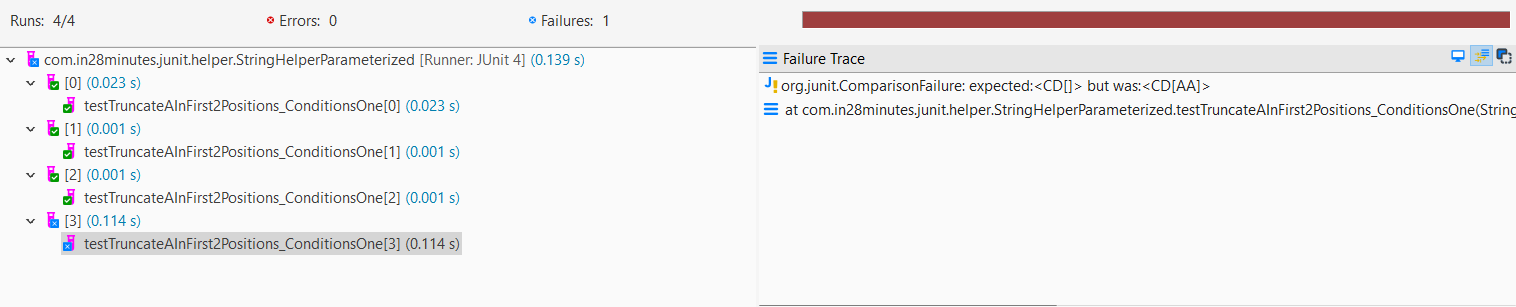
}

}





So 1 @Parameters method but **4 tests**



they are shown using indexes, here 3 passed and 1 failed

\*\*\* For another Parametrized test , use a separate class is must

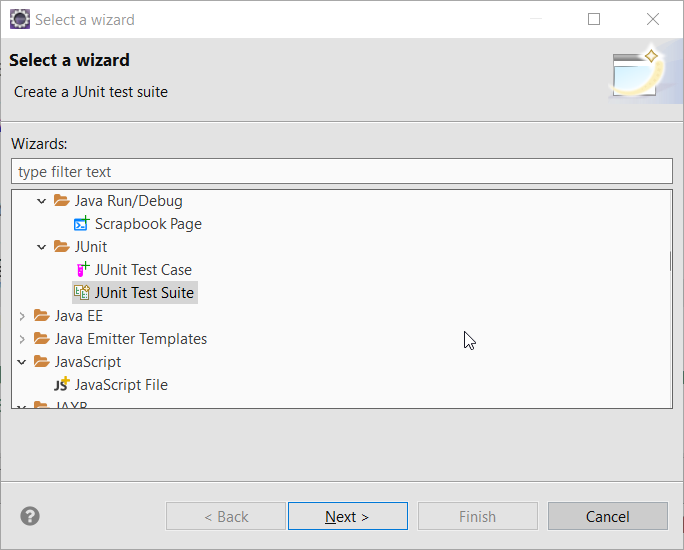
### Test Suites

To organize our test we use test suite

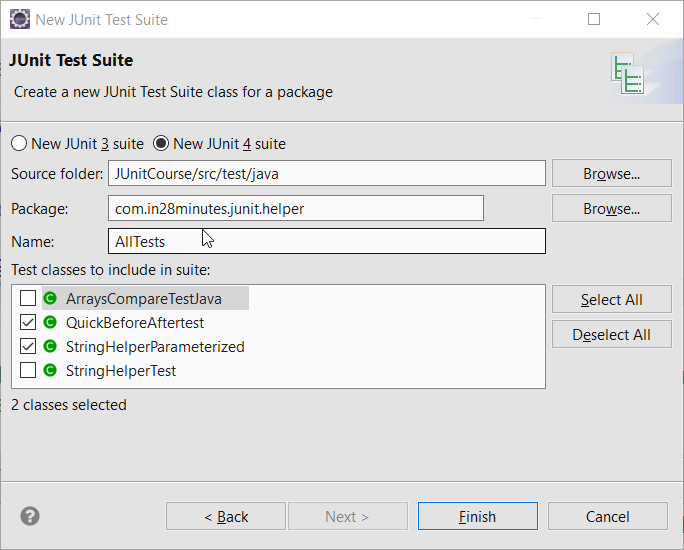
If we do right click on a package and run as J Unit test, it will execute all the test in the package, to get control over it we will use test suite.

**Steps:**

1. Right click on package >>> J Unit test suite



1. Select the desired test cases of the package



package com.in28minutes.junit.helper;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

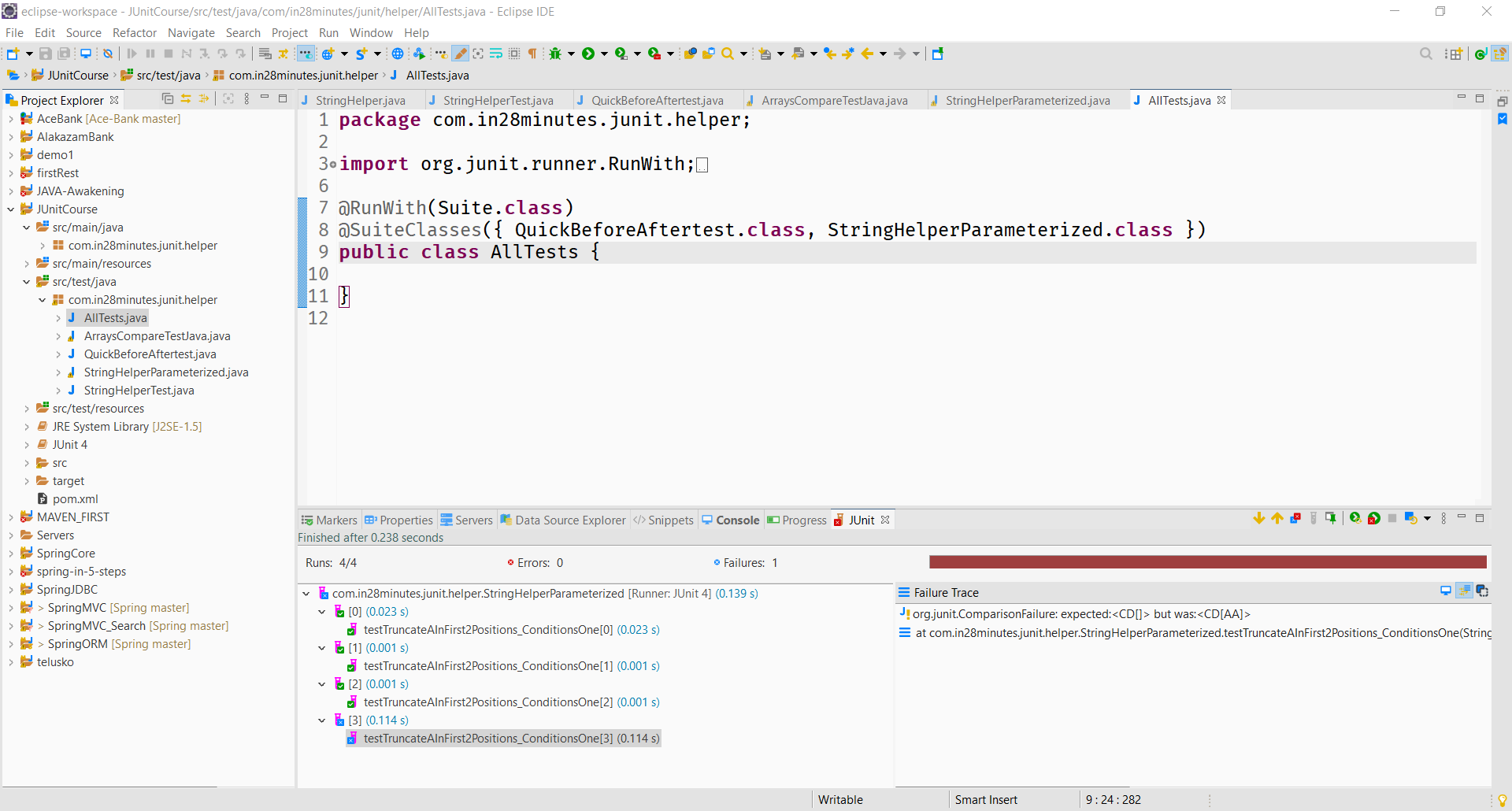
import org.junit.runners.Suite.SuiteClasses;

**@RunWith**(Suite.class)

**@SuiteClasses**({ QuickBeforeAftertest.class, StringHelperParameterized.class })

public class AllTests {

}



(Suite.class) **@SuiteClasses ->** Array of different classes

*Suites gives flexibility to select our desired test classes and increase the efficiency*

[MOCKITO](https://github.com/in28minutes/MockitoTutorialForBeginners)

*https://github.com/in28minutes/MockitoTutorialForBeginners*

## *Gain expertise on the most popular java mocking framework*

* [Installing Eclipse, Maven and Java](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "installing-tools)
* [Running Examples](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "running-examples)
* [Course Overview](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "course-overview)
  + [Course Steps](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "step-list)
  + [Expectations](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "expectations)
* [About in28Minutes](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "about-in28minutes)
  + [Our Beliefs](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "our-beliefs)
  + [Our Approach](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "our-approach)
  + [Find Us](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "useful-links)
  + [Other Courses](https://github.com/in28minutes/MockitoTutorialForBeginners" \l "other-courses)

## Installing Tools

* PDF : <https://github.com/in28minutes/SpringIn28Minutes/blob/master/InstallationGuide-JavaEclipseAndMaven_v2.pdf>
* Video : <https://www.youtube.com/watch?v=eqRF4xHoGck>

## Course Overview

#### Mockito

* Step 01 : Set up an Eclipse Project with JUnit and Mockito frameworks. First Green Bar.
* Step 02 : Example to start understanding why we need mocks.
* Step 03 : What is a stub? Create an unit test using Stub? Disadvantages of Stubs.
* Step 04 : Your first Mockito code! Hurrah!!! Lets use Mockito to mock TodoService.
* Step 05 : Stubbing variations with Mockito. A few mockito examples mocking List class : Multiple return values, Argument Matchers and throwing exceptions.
* Some Theory : Mockito vs EasyMock <https://github.com/mockito/mockito/wiki/Mockito-vs-EasyMock>
* Step 06 : Introduction to BDD. Given When Then. BDD Mockito Syntax.
* Step 07 : How to verify calls on a mock? Verify how many times a method is called. We will add deleteTodo method to the TodoService.
* Step 08 : How to capture an argument which is passed to a mock?
* Step 09 : Hamcrest Matchers.
* Step 10 : Let's simplify things with Mockito Annotations. @Mock, @InjectMocks, @RunWith(MockitoJUnitRunner.class), @Captor
* Step 11 : JUnit Rules. Using MockitoJUnit.rule() instead of @RunWith(MockitoJUnitRunner.class).
* Step 12 : Real world Example with Spring
* Step 13 : What is a spy? How to spy with Mockito?
* Step 14 : Some Theory : Why does Mockito not allow stubbing final and private methods?
* Step 15 : Using PowerMock and Mockito to mock a Static Method.
* Step 16 : Using PowerMock and Mockito to invoke a private Method.
* Step 17 : Using PowerMock and Mockito to mock a constructor.
* Step 18 : Good Unit Tests.

### Expectations

* You should know Java.
* You are NOT expected to have any experience with Eclipse or Maven.
* We will help you install Eclipse and get up and running with Maven.

### Running Examples

* Download the zip or clone the Git repository.
* Unzip the zip file (if you downloaded one)
* Open Command Prompt and Change directory (cd) to folder containing pom.xml
* Open Eclipse
  + File -> Import -> Existing Maven Project -> Navigate to the folder where you unzipped the zip
  + Select the right project
* Choose the Spring Boot Application file (search for @SpringBootApplication)
* Right Click on the file and Run as Java Application
* You are all Set

*Step 1*

Setup the Project