in28minutes

Unit Testing with JUnit 5

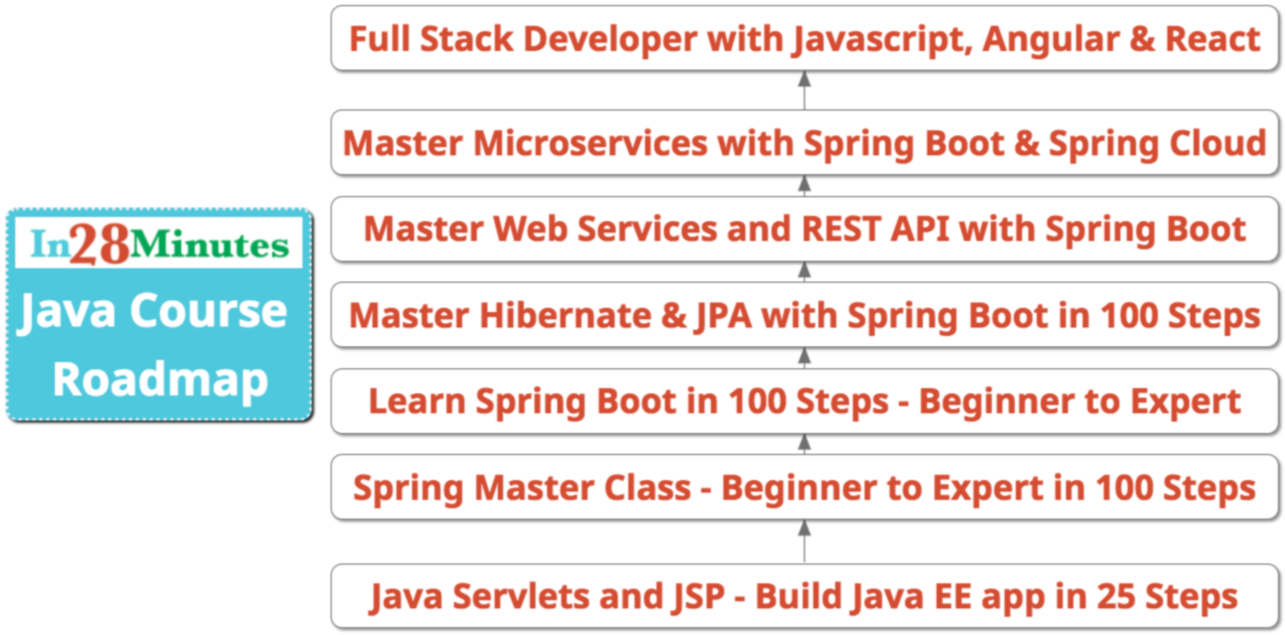
Learn to write awesome unit tests with JUnit 5 in 20 Easy Steps!

Table of Contents

Congratulations

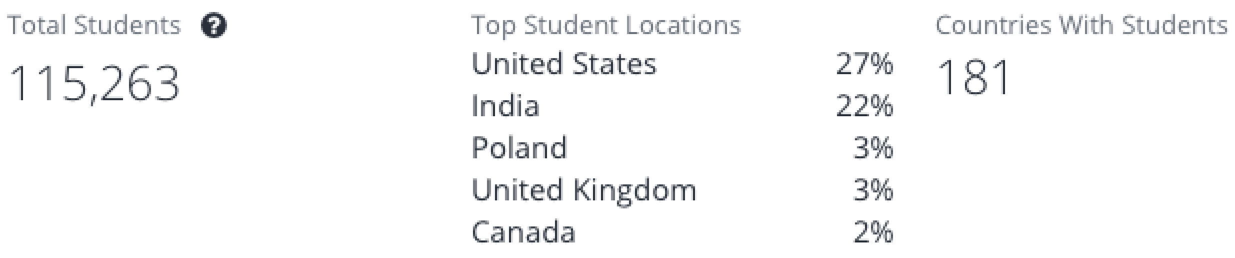
You have made a great choice in learning with in28Minutes. You are joining 150,000+ Learners learning everyday with us.

150,000+ Java beginners are learning from in28Minutes to become experts on APIs, Web Services and Microservices with Spring, Spring Boot and Spring Cloud.


In28Minutes
Java Course Roadmap
Full Stack Developer with Javascript, Angular & React
Master Microservices with Spring Boot & Spring Cloud
Master Web Services and REST API with Spring Boot
Master Hibernate & JPA with Spring Boot in 100 Steps
Learn Spring Boot in 100 Steps - Beginner to Expert
Spring Master Class - Beginner to Expert in 100 Steps
Java Servlets and JSP - Build Java EE app in 25 Steps

About in28Minutes

How did in28Minutes get to 150,000 learners across the world?



Total Students
115,263

Top Student Locations	
United States	27%
India	22%
Poland	3%
United Kingdom	3%
Canada	2%

Countries With Students
181

> *We are focused on creating the awesome course (learning) experiences. Period.*

An awesome learning experience?

What's that?

You need to get insight into the in28Minutes world to answer that.

You need to understand "***The in28Minutes Way***"

- What are our beliefs?
- What do we love?
- Why do we do what we do?
- How do we design our courses?

Let's get started on "***The in28Minutes Way***"!

Important Components of "The in28Minutes Way"

- Continuous Learning
- Hands-on
- We don't teach frameworks. We teach building applications!
- We want you to be strong on the fundamentals

- Step By Step

- Efficient and Effective

- Real Project Experiences
- Debugging and Troubleshooting skills
- Modules - Beginners and Experts!
- Focus on Unit Testing
- Code on Github

- Design and Architecture
- Modern Development Practices
- Interview Guides
- Bring the technology trends to you
- Building a connect
- Socially Conscious
- We care for our learners
- We love what we do

Troubleshooting Guide

We love all our 150,000 learners. We want to help you in every way possible.

We do not want you to get stuck because of a simple error.

This 50 page troubleshooting guide and faq is our way of thanking you for choosing to learn from in28Minutes.

.in28Minutes Trouble Shooting Guide

Getting Started

Recommended Versions

Tool/Framework/Language	Recommended Version	More Details
Java	Java 8	http://www.in28minutes.com/spr...
Eclipse	Eclipse Java EE Oxygen	Basics

Installation

- Video : https://www.youtube.com/playlist?list=PLBBog2r6uMCSmMVTW_QmDLyASBvovyAO3
- PDF : https://github.com/in28minutes/SpringIn28Minutes/blob/master/InstallationGuide-JavaEclipseAndMaven_v2.pdf
- More Details : https://github.com/in28minutes/getting-started-in-5-steps

Troubleshooting

- A 50 page troubleshooting guide with more than 200 Errors and Questions answered

Unit Testing with JUnit 5 - Course Overview

Github Page :

https://github.com/in28minutes/spring-unit-testing-with-junit-and-mockito/blob/master/junit-5-course-guide.md

Overview

JUnit is most popular Java Unit Testing Framework. The new version of JUnit - Junit 5 or Jupiter is even more special.

In this course, we look into the important features of JUnit 5. We understand the need for unit testing and learn how to write great unit tests with JUnit 5.

JUnit is a unit testing framework for the Java programming language.

In this Beginners Tutorial on JUnit 5, you will learn how to

- Create a new project for JUnit Tests
- Create and Run JUnit Tests
- Write Good Unit Tests
- Use assert methods
- Use basic JUnit Annotations - @Test, @BeforeEach, @AfterEach, @AfterAll, @BeforeAll
- Test Performance and Exceptions in Unit Tests
- Write Parameterized Tests
- Adhere to JUnit Best Practices
- Use Eclipse to write and run JUnit Tests

Step By Step Details

- Step 01 - Introduction to Unit Testing - Test Pyramid
- Step 02 - First Junit Test - Red bar
- Step 03 - Absence of failure is success
- Step 04 - First Unit Test with JUnit - String length() method
- Step 05 - Writing JUnit Assertions - assertNull and assertTrue
- Step 06 - Writing Assertions for Arrays - assertArrayEquals
- Step 07 - Setting up data for every test - @BeforeEach, @AfterEach
- Step 08 - Setting up database connections - @BeforeAll, @AfterAll
- Step 09 - Tip - Testing Exceptions with JUnit
- Step 10 - Tip - @DisplayName and test methods need not be public
- Step 11 - Basics of Parameterized tests
- Step 12 - Advanced Paramaterized Tests with Csv Source
- Step 13 - Tip - Giving a name to a Parameterized Test
- Step 14 - Tip - Repeat same test multiple times
- Step 15 - Tip - Unit Testing for Performance
- Step 16 - Tip - Disable Unit Tests
- Step 17 - Tip - Grouping Tests with @Nested
- Step 18 - Tip - JUnit 5 = Platform + Jupiter + Vintage
- Step 19 - Tip - JUnit 4 vs JUnit 5
- Step 20 - Tip - JUnit Best Practices
- Step 21 - Tip - JUnit Patterns - http://xunitpatterns.com

Exercises

- Step 04 - First Unit Test with JUnit - String length() method
  - Write unit test for Math.min and Math.max methods!
- Step 05 - Writing JUnit Assertions - assertNull and assertTrue
  - Exercise : Write assertEquals for two other data types
- Step 06 - Writing Assertions for Arrays - assertArrayEquals
  - Spot the bug : Reverse expected and actual values
- Step 11 - Basics of Parameterized tests
  - Write unit tests using ints attribute of Parameterized Test for a Math method.

Unit Testing with JUnit 5 - Complete Code

JUnit 5 Complete Code Example

```
package com.in28minutes.junit5;

import static
org.junit.jupiter.api.Assertions.assertArrayEquals;
import static
org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static
org.junit.jupiter.api.Assertions.assertNotNull;
import static
org.junit.jupiter.api.Assertions.assertThrows;
import static
org.junit.jupiter.api.Assertions.assertTimeout;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.time.Duration;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInfo;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import org.junit.jupiter.params.provider.ValueSource;

class StringTest     {

        private String str;

        @BeforeAll
        static void beforeAll() {
                System.out.println("Initialize connection
to database");
        }

        @AfterAll
        static void afterAll() {
                System.out.println("Close connection to
database");
        }

        @BeforeEach // @Before
        void beforeEach(TestInfo info) {
                System.out.println("Initialize Test Data
for  " + info.getDisplayName());
        }

        @AfterEach // @After
        void afterEach(TestInfo info) {
                System.out.println("Clean up Test Data for
" + info.getDisplayName());
        }
        @Test
        @Disabled // @Ignored
        void length_basic()

 {
                int actualLength = "ABCD".length();
                int expectedLength = 4;
                assertEquals(expectedLength,
actualLength);
                // Assert length == 4
                // Write test code
                // Invoke method square(4) => CUT
                // Checks in place - 16 => Assertions
        }

        @Test
        void length_greater_than_zero() {
                assertTrue("ABCD".length() > 0);
                assertTrue("ABC".length() > 0);
                assertTrue("A".length() > 0);
                assertTrue("DEF".length() > 0);
        }

        @ParameterizedTest
        @ValueSource(strings = { "ABCD", "ABC", "A", "DEF"
})
        void
length_greater_than_zero_using_parameterized_test(String
str) {
                assertTrue(str.length() > 0);
        }

        @ParameterizedTest(name = "{0} toUpperCase is {1}")
        @CsvSource(value = { "abcd, ABCD", "abc, ABC",
"'','''", "abcdefg, ABCDEFG" })
        void uppercase(String word, String capitalizedWord)
 {
                assertEquals(capitalizedWord,
word.toUpperCase());
        }


        @ParameterizedTest(name = "{0} length is {1}")

        @CsvSource(value = { "abcd, 4", "abc, 3", "'',0",
"abcdefg, 7"  })
        void length(String word, int expectedLength) {
                assertEquals(expectedLength,
word.length());
        }

        @Test
        @DisplayName("When length is null, throw an
exception")
        void length_exception() {

                String str = null;

                assertThrows(NullPointerException.class,

                                () -> {
                                        str.length();
                                }

                );
        }

        @Test
        @Disabled
        void performanceTest() {
                assertTimeout(Duration.ofSeconds(5), () ->
{
                        for (int i = 0; i <= 1000000; i++)
 {
                                int j = i;
                                System.out.println(j);
                        }


        }

                );

        }

        @Test
        void toUpperCase_basic() {
                String str = "abcd";
                String result = str.toUpperCase();
                assertNotNull(result);
                //  assertNull(result);
                assertEquals("ABCD", result);
        }

        @Test
        @RepeatedTest(1)
        void contains_basic() {
                assertFalse("abcdefgh".contains("ijk"));
        }

        @Test
        void split_basic() {
                String str = "abc def ghi";
                String actualResult[] = str.split(" ");
                String[] expectedResult = new String[] {
"abc", "def", "ghi" };
                assertArrayEquals(expectedResult,
actualResult);
```

```
    }


    @Nested
    @DisplayName("For an empty String")
    class EmptyStringTests  {



    @BeforeEach
            void setToEmpty() {
                    str = "";


    }


            @Test
            @DisplayName("length should be zero")
            void lengthIsZero() {
                    assertEquals(0,str.length());
            }

            @Test
            @DisplayName("upper case is empty")
            void uppercaseIsEmpty() {
                    assertEquals("",str.toUpperCase());
            }

    }

    @Nested
    @DisplayName("For large strings")
    class LargeStringTests {

            @BeforeEach
            void setToALargeString() {
                    str =
 "Afsjdjfljsadfkljsadlkfjlajbvjcxzbnhrewu";


        }


                @Test
                void test()  {

                }


        }
}
```

References

References

- Intellij
  - https://www.jetbrains.com/help/idea/configuring-testing-libraries.html
  - https://blog.jetbrains.com/idea/2016/08/using-junit-5-in-intellij-idea/

- Functional Programming - https://youtu.be/aFCNPHfvqEU
- JUnit - https://junit.org/junit5/docs/current/user-guide/
- Good Unit Testing
  - FIRST. https://pragprog.com/magazines/2012-01/unit-tests-are-first
  - Patterns - http://xunitpatterns.com

- More Advanced Stuff
  - AssertJ - https://joel-costigliola.github.io/assertj/
  - Mockito - https://github.com/mockito/mockito/wiki/FAQ
  - JsonPath - https://github.com/json-path/JsonPath
  - Setting up JUnit 5 with Mockito and Spring Boot 2.0 - https://medium.com/@dSebastien/using-junit-5-with-spring-boot-2-kotlin-and-mockito-d5aea5b0c668