Cognizant | Delivery Excellence

JUnit

Bhaskar D

May 2021

Agenda

Cognizant

What is JUnit?

- JUnit is an open source Unit Testing Framework for Java programming language. It is useful for Java Developers to write and run repeatable tests. It is a family of unit testing frameworks collectively known as xUnit.

- JUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

JUnit Advantages and uses

- ❖ Programmers become more productive
- ❖ Increase the quality of the developed code
- ❖ Enables writing test cases while developing the software that helps test early and detect issues.
- ❖ Ensure the functionality is performing as expected every time the code is modified by the use of repeatable automated test cases.
- ❖ Supported by all IDE including Eclipse, Netbeans, RAD etc.
- ❖ Integrates with Ant and Maven that enables execution of test suites or test cases as part of the build process, capturing test result and reporting.
- ❖ Widely adopted by many organizations around the world for performing unit testing in Java programming language

JUnit Features

- JUnit is an open source framework, which is used for writing and running tests.
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit is elegantly simple. It is less complex and takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.

Cognizant

Setup JUnit

Tool Name	JUnit version 4 or 5
Software Requirements	JDK 1.5 or above
Hardware requirements	Windows7/Linux/Mac*
Other requirements	Eclipse Mars

Processor	1.5 GHz
RAM	1 GB
Disk space	5 GB

* JUnit will be executed under Eclipse, provided Eclipse IDE system requirement

Cognizant

Setup JUnit Cont…

Step 1: Verify Java Installation

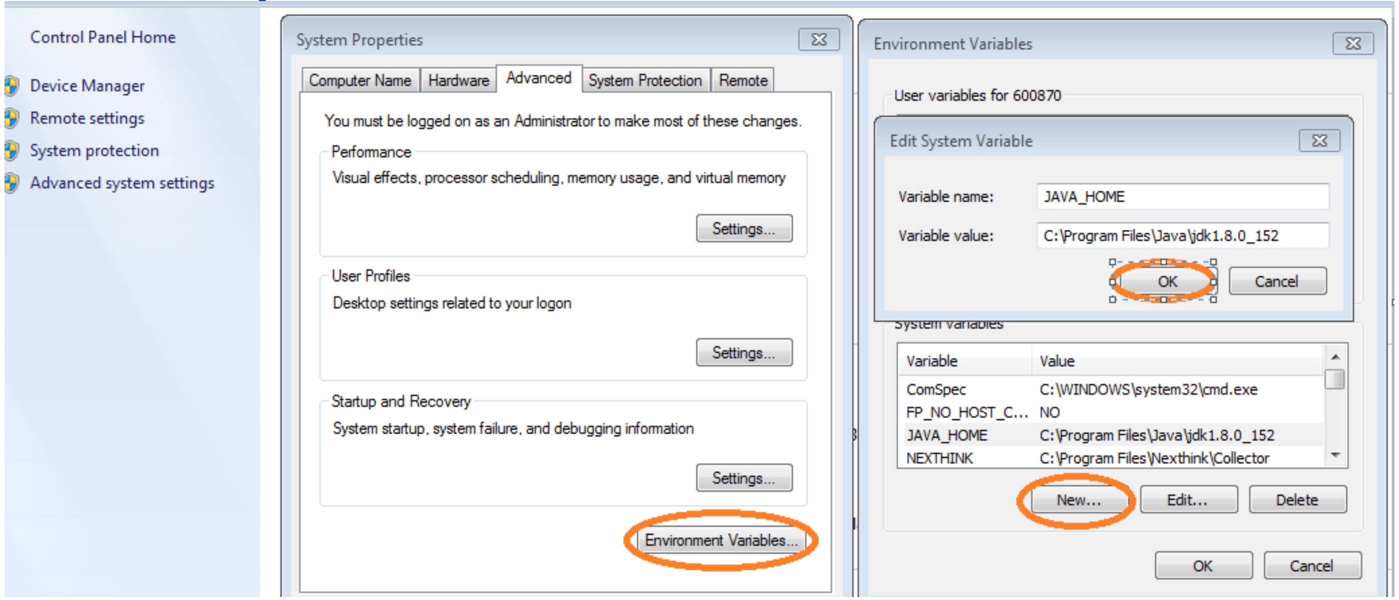
OS	Command	Output
Windows	c:\>java -version	java version "1.8.0_251" Java(TM) SE Runtime Environment (build 1.8.0_251-b08) Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
Linux	$java -version	java version "1.8.0_251" Java(TM) SE Runtime Environment (build 1.8.0_251-b08) Java HotSpot(TM) Client VM (build 25.251-b08, mixed mode, sharing)
Mac	~ $ java -version	java version "1.8.0_251" Java(TM) SE Runtime Environment (build 1.8.0_251-b08) Java HotSpot(TM)64-Bit Server VM (build 251-b08, mixed mode, sharing)

Cognizant

Setup JUnit Cont…

Step 2: Set JAVA Environment

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine


Control Panel Home

Device Manager

Remote settings

System protection

Advanced system settings

System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

Settings...

User Profiles

Desktop settings related to your logon

Settings...

Startup and Recovery

System startup, system failure, and debugging information

Settings...

Environment Variables...

Environment Variables

User variables for 600870

Edit System Variable

Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.8.0_152

OK Cancel

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...	NO
JAVA_HOME	C:\Program Files\Java\jdk1.8.0_152
NEXTHINK	C:\Program Files\Nexthink\Collector

New... Edit... Delete

OK Cancel

Setup JUnit Cont…

Step 3: Download JUnit Archive

Download the latest JUnit jar file from https://sourceforge.net/projects/junit/files/junit/4.10/ and copied it into your local path.

Step 4: Set JUnit Environment

Set the **JUNIT_HOME** environment variable to point to the base directory location where JUNIT jar is stored on your machine.

Step 5: Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the JUNIT jar location.

Annotations and Assert Statements

ANNOTATIONS is a special form of syntactic meta-data that can be added to Java source code for better code readability and structure

Sl No	Annotations	Description
1	@Test	The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.
2	@Before	This annotation is used if you want to execute some statement such as preconditions before each test case.
3	@BeforeClass	Annotating a public static void method with @BeforeClass causes it to be run once before any of the test methods in the class.
4	@After	If you allocate external resources in a Before method, you need to release them after the test runs. Annotating a public void method with @After causes that method to be run after the Test method.
5	@AfterClass	This will perform the method after all tests have finished. This can be used to perform clean-up activities.
6	@Ignore	The Ignore annotation is used to ignore the test and that test will not be executed.

Annotations and Assert Statements

Sl No	Methods	Description
1	void assertEquals(boolean expected, boolean actual)	Checks that two primitives/objects are equal.
2	void assertTrue(boolean condition)	Checks that a condition is true.
3	void assertFalse(boolean condition)	Checks that a condition is false.
4	void assertNotNull(Object object)	Checks that an object isn't null.
5	void assertNull(Object object)	Checks that an object is null.
6	void assertSame(object1, object2)	The assertSame() method tests if two object references point to the same object.
7	void assertNotSame(object1, object2)	The assertNotSame() method tests if two object references do not point to the same object.
8	void assertArrayEquals(expectedArray, resultArray);	The assertArrayEquals() method will test whether two arrays are equal to each other.

Assert is a method useful in determining Pass or Fail status of a test case, In Junit all the assertions are in Assert class

JUnit 4 vs 5

JUnit 5 vs JUnit 4 – Others

FEATURE	JUnit 4	JUnit 5
JDK Version	Min 1.5	Min 1.8
Tagging and Filtering	@category annotation is used.	@tag annotation is used.

Test Suites	@RunWith and @Suite annotation.
	import org.junit.runner.RunWith;
	import org.junit.runners.Suite;
	 @RunWith(Suite.class)
	@Suite.SuiteClasses({
	        ExceptionTest.class,
	        TimeoutTest.class
	})
		@RunWith, @SelectPackages and @SelectClasses e.g.
		import org.junit.platform.runner.JUnitPlatform;
		import org.junit.platform.suite.api.SelectPackages;
		import org.junit.runner.RunWith;
		 @RunWith(JUnitPlatform.class)
		@SelectPackages("com.howtodoinjava.junit5.examples")

3rd Party Integration	There is no integration support for 3rd party plugins and IDEs	Dedicated sub-project for this purpose i.e. JUnit Platform. It defines the TestEngine API for developing a testing framework that runs on the platform.

JUnit 4 vs 5

JUnit 5 vs JUnit 4 – Annotations

FEATURE	JUnit 4	JUnit 5
Execute before all test methods in the current class	@BeforeClass	@BeforeAll
Execute after all test methods in the current class	@AfterClass	@AfterAll
Execute before each test method	@Before	@BeforeEach
Execute after each test method	@After	@AfterEach
Disable a test method / class	@Ignore	@Disabled
Test factory for dynamic tests	NA	@TestFactory
Nested tests	NA	@Nested
Tagging and filtering	@Category	@Tag
Register custom extensions	NA	@ExtendWith

Cognizant

JUnit Annotations

Annotations	Description
@BeforeAll	Executes a method before all tests.
@AfterAll	Executes a method after all tests
@BeforeEach	Execute before each test method
@AfterEach	Execute after each test method,
@Disabled	This is used to disable or skip tests at class or method level.
@TestFactory	Used to signal that the annotated method is a test factory method.
@Nested	Can be used to mark a nested class to be included in the test cases.
@Order	Is useful when we want to create a test pack with selected tests.
@RepeatedTest	Ability to repeat a test a specified number of times

JUnit Test Framework

JUnit test framework provides the following important features:

- Fixtures
  - Fixtures is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well-known and fixed environment in which tests are run so that results are repeatable. It includes:
    - setUp() method, which runs before every test invocation.
    - tearDown() method, which runs after every test method.
- Test suites
  - A test suite bundles a few unit test cases and runs them together. In JUnit, both @RunWith and @Suite annotation are used to run the suite test.
- Test runners
  - Test runner is used for executing the test cases.
- JUnit classes
  - JUnit classes are important classes, used in writing and testing JUnits. Some of the important classes are:
    - Assert - Contains a set of assert methods.
    - TestCase - Contains a test case that defines the fixture to run multiple tests.
    - TestResult - Contains methods to collect the results of executing a test case.

Exceptions

- ❖ JUnit provides the facility to trace the exception and also to check whether the code is throwing expected exception or not.
- ❖ Junit4 provides an easy and readable way for exception testing, you can use
  - ✓ Optional parameter (expected) of @test annotation and
  - ✓ To trace the information ,"fail()" can be used
- ❖ While Testing exception, you need to ensure that exception class you are providing in that optional parameter of @test annotation is the same. This is because you are expecting an exception from the method you are Unit Testing, otherwise our JUnit test would fail.

JUnit Dynamic Test

What is DynamicTest?

The standard tests annotated with @Test annotation are static tests which are fully specified at the compile time. A DynamicTest is a test generated during runtime. These tests are generated by a factory method annotated with the @TestFactory annotation.

The DynamicTests are executed differently than the standard @Tests and do not support lifecycle callbacks. Meaning, the @BeforeEach and the @AfterEach methods will not be called for the DynamicTests.

.

JUnit Code Coverage

What is Code Coverage?

Code coverage means measuring how much of your code is executed during your unit tests. Basically, that means that after running your unit tests, you get a report showing you how many percent of the code that was executed during the tests, and also what lines precisely that were executed.

To measure code coverage you need a coverage tool. List of code coverage tools for Java:

- ❑ Jacoco
- ❑ EclEmma
- ❑ Emma
- ❑ Cobertura

.

JUnit – Best Practices

- Test only one code unit at a time
- Don’t make unnecessary assertions
- Make each test independent of all the others
- Don’t unit-test configuration settings
- Name your unit tests clearly and consistently
- All methods, regardless of visibility, should have appropriate unit tests
- Put assertion parameters in the proper order
- Ensure that test code is separated from production code
- Do not print anything out in unit tests

Thank You

Bhaskar D

bhaskar.dara@cognizant.com

Cognizant | Delivery Excellence