**JUNIT Framework (version-Junit4.11)**

**Current version – 4.12**

* It is a unit testing framework for java programmes which is an important test driven development for java developers to write and run repeatable tests.
* Developers write and execute unit test first before writing any code. Once after writing the code the developers should execute all the test cases and it should pass and this should be followed every time you add the code.
* In this, we are going to test a unit which is a single method/group of methods/set of classes – to check whether the given conditions are pass/fail according to the given input.
* It is an instance of **Xunit** architecture.

**Junit.framework.TestCase - JUnit 3.8 and earlier**

**org.junit.Test – Junit 4 and later**

* With JUnit 4 you are more capable of identifying exception. You can define expected exception as a parameter while using @test annotation.
* Parameterized test is introduced, which enables us to use parameters.
* JUnit4 still can execute JUnit3 tests.
* JUnit 4 can be used with java5 or higher version.
* While using JUnit4, you are not required to extend JUnit.framework.TestCase. You can just create a simple java class.
* You need to use annotations in spite of special method name as before.
* Instead of using setup method, you need to use @before annotation.
* Instead of using teardown method, put @after annotation.

Instead of using testxxxx before method name, use @test annotation

**Link** - <http://www.asjava.com/junit/junit-3-vs-junit-4-comparison/>

**Why we need Junit:**

* It finds bugs early in the code, which makes our code more reliable.
* JUnit is useful for developers, who work in a test-driven environment.
* Unit testing forces a developer to read code more than writing.
* You develop more readable, reliable and bug-free code which builds confidence during development.

**Advantages**

* JUnit addresses the above issues, It’s used to test an existing class. Suppose, If we have Calculation class then, we need to write CalculationTest to test the Calculation class.
* Using JUnit we can save testing time.
* In real time, in the web application development we implement JUnit test cases in the DAO classes. DAO classes can be tested with out server.
* With JUnit we can also test Struts / Spring applications but most of the time we test only DAO classes. Even some times service classes also tested using JUnit.
* Using JUnit, If we want to test the application (for web applications) then server is not required so the testing becomes fast.

**Implementation:**

**Installing Junit**

1. **Download jar file and add in the project through external libraries**

<http://junit.org/junit4/>

1. **Create java project -> right click on it -> go to new and select Junit 4**
2. **If it is a maven project then add in the pom.xml file**

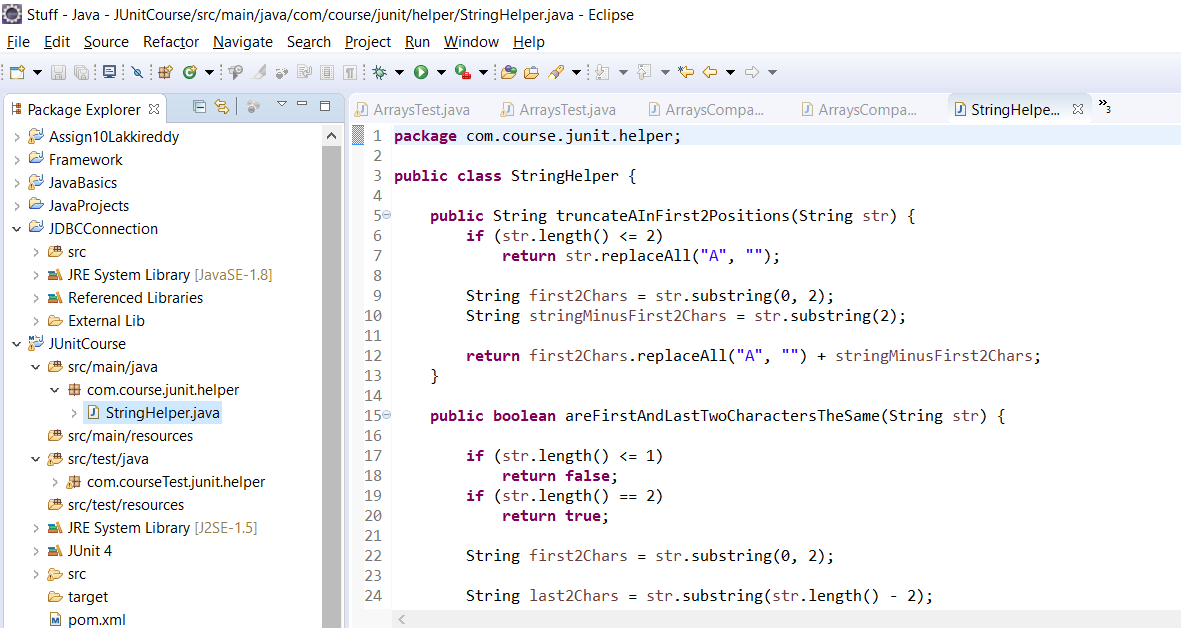
<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<name>JUnit</name>

Create a Maven project and then in the src/main/java paste the java class file – **StringHelper**



and then if you want to do unit testing we have to test the java class file in **src/test/java**.

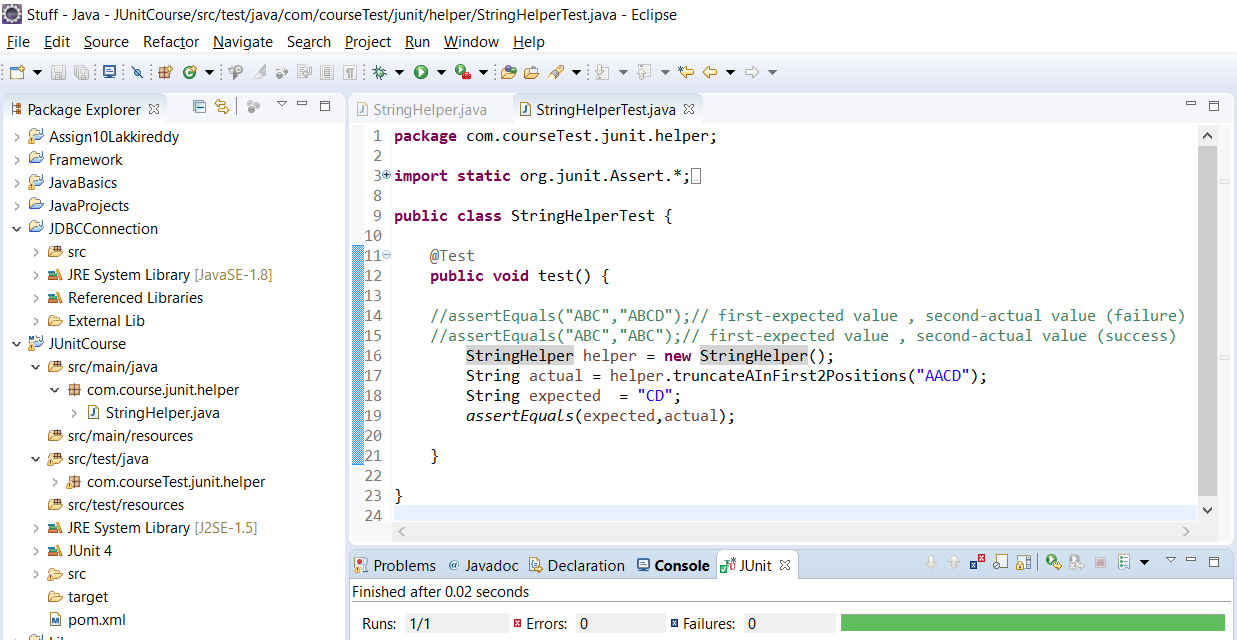
Go to src/test/java -> right click on it and select new and click on Junit Test case -> browse the required java class which is going to be tested (String Helper)

Click on Ok -> then write package name which is given for the class and write name as StringHelperTest and click on Finish - > it prompts you to add Junit 4 -> click on ok -> your first Junit test case is ready.

Now, the test class is created.

**@Test** - It is an annotation which is defined from org.junit.Test package.

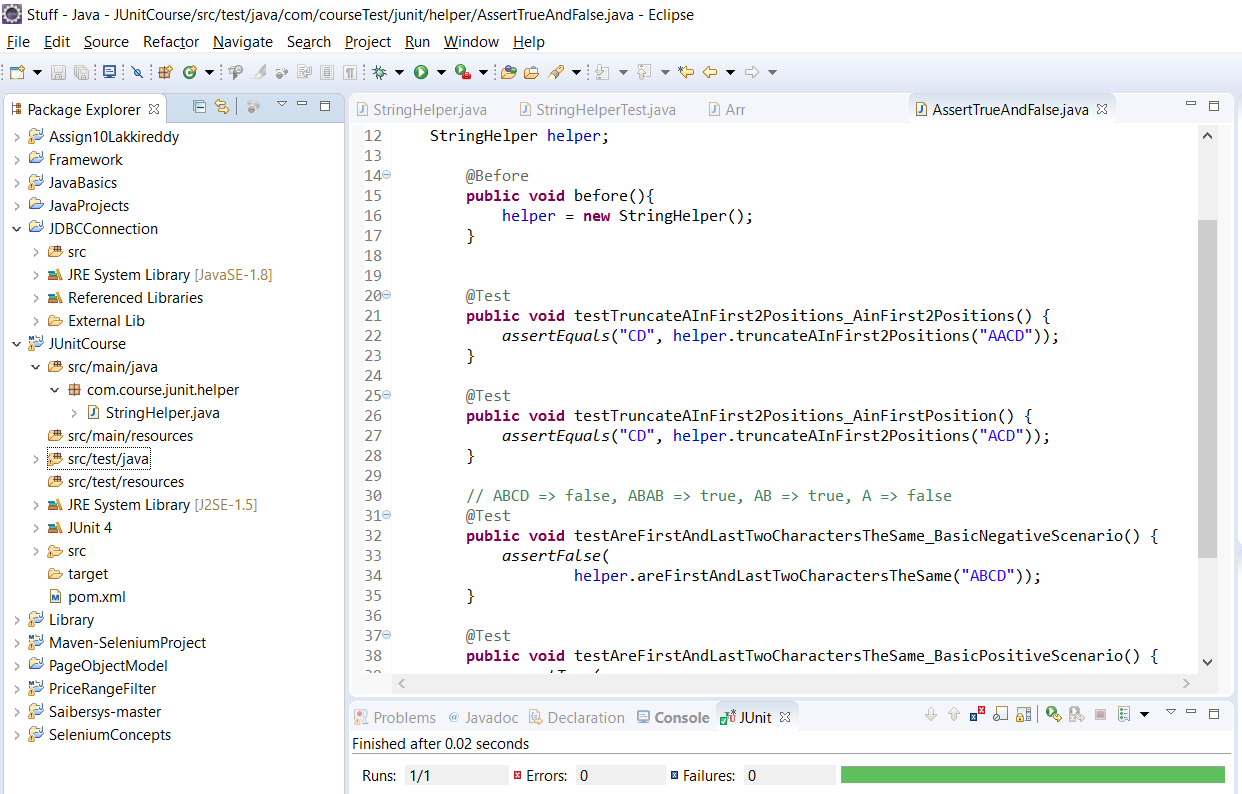
* Here, we are comparing the actual and expected values and we got success after running the test if they are equal or else failure.



**AssetTrue and AssertFalse:**

It checks for 2 conditions:

1. If given conditions are true when the value in assetTrue is correct and it gives success (or) it gives failure when the value is not correct.
2. If given conditions are true when the value in assetFalse is not correct and it gives success (or) it gives failure when the value is correct.



**Annotations:**

**@Before:** it will be executed before each test

**@BeforeClass**: it will be executed before the specified number of tests. For instance, if @Before has 10 tests then it is done 10 times and @BeforeClass is done only once.

**@After:** it will be executed after each test

**@AfterClass:** it will be executed after the specified number of tests. For instance, if @After has 10 tests then it is done 10 times and @AfterClass is done only once.

