# JUnit

Junit is a unit testing framework for java programming language. It plays a critical role in test-driven development, and a family of unit testing framework collectively known as xUnit. Junit promotes “first testing and then coding”. This approach is like “test little and code little”.



Features of JUnit.

- JUnit is an open source framework.
- Provides annotations to identify test method
- Provides assertions for testing expected results.
- Provides test runners for running test.
- It allows to write code faster which increases qulity.
- It is very simple and takes less time.
- It automatically runs and they check their own results and provide immediate feedback.
- It can be organized to test suits which contains test cases and other test suits.

Basic example for JUnit.

```
public class Addition {
	public int add(int a, int b) {
		return a + b;
	}
}
```

```java
public class AdditionTest {
	@Test
	public void addIntTest() {
		Addition addition = new Addition();
		int sum = addition.add(20, 30);
		Assert.assertEquals(50, sum);
	}
}

public class TestRunner {
	public static void main(String[] args) {
		Result result = JUnitCore.runClasses(TestCases.class);

		List<Failure> failures = result.getFailures();
		for (Failure failure : failures) {
			System.out.println(failure.getMessage());
		}

		System.out.println(result.wasSuccessful());
	}
}
```

API

The most important package in JUnit is org.junit, which contains all the core classes. Some important classes are Assert, TestCase, Result, and Suite

org.junit.Assert

This class provides a set of assertion methods useful for writing tests. Only failed assertion will be recorded

Method  in Assert

- assertEquals(Object, Object) : void
- assertEquals(dataType,dataType) : void
- assertEqualsArray(array, array) : void

- assertNotNull(Object) : void
- assertTrue(Boolean Expression) : void
- assertNull(Object) : void
- fail() : void

org.juint.TestCase

This is a sub class of Assert and implementation class of Test

Methods in TestCase

- countTestCases() : int
- createResult() : TestResult
- getName() : String
- run() : TestResult
- run(TestResult) : void
- setName(String) : void
- setup() : void
- teardown() : void

Annotations

They are meta-tags that you can add to your code. The annotation in JUnit provide the following information about test methods

- which methods are going to run before and after test method
- which methods run before and after all the methods
- which methods are test method
- which methods should be ignored.

@Test

This tells JUnit that the public void method to which it is attached can be run as a test case.

@Before

Annotating a public void method with this causes that method to be run before each test case.

@After

Annotating a public void method with this causes that method to be run after each test case.

@BeforeClass

Annotating a public static void method with this causes that method to be run once before test case of a class.

@AfterClass

Annotating a public static void method with this causes that method to be run once after test case of a class.

@Ignore

This annotation is used to ignore the test and that test will not be executed.

@RunWith and @SuiteClasses

These annotations are used to run the suite tests. These helps to bundle few test cases and run together.

## Time Test

JUnit provides a timeout for a test case. For @Test annotation add timeout parameter to test time for the test case.

## Exception Test

Junit provides an option of tracing the exception in a code. For @Test annotation add expected parameter and specify an exception type to test whether the code throws exception or not

## Parameterized Test

JUnit 4 has introduced new feature called parameterized test. This allow a developer to run the same test over and over again using different values. There are 5 steps to follow to create a parameterized test.

- Annotate the test class with @RunWith(Parameterized.class).
- Create a public static method annotated with @Parameters that returns collection of array of objects as test data set.
- Create public constructor that takes in what is equivalent to one row of test data.

# Example for parameterized tests

```
@RunWith(Parameterized.class)
public class AdditionTest {
	private int a;
	private int b;
	private int sum;
	public AdditionTest(int a, int b, int sum) {
		this.a=a;
		this.b=b;
		this.sum=sum;
	}
	@BeforeClass
	public static void executeBefore() {
		System.out.println("Testing Addition starting");
	}
	@Before
	public void executeBeforeTestCase() {
		System.out.println("Test method starting");
	}
	@Test
	public void addIntTest() {
		Addition addition = new Addition();
		int sum = addition.add(a, b);
		Assert.assertEquals(this.sum, sum);
	}
	@After
	public void executeAfterTestCase() {
		System.out.println("Test methods ending");
	}
	@AfterClass
	public static void executeAfter() {
		System.out.println("Testing Addition ending");
	}
	@Parameters
	public static Collection<Object[]> inputsAndOutputs() {
		return Arrays.asList(new Object[][] {
				{1,2,3},
				{5,5,10},
				{7,3,10},
				{-4,5,1}
		});
	}
}
```

# Example for suite test cases

```
public class AdditionTest {
	@Test
	public void addIntTest() {
		Addition addition = new Addition();
		TestCase.assertEquals(50, addition.add(20, 30));
	}
	@Test
	public void addDoubleTest() {
		Addition addition = new Addition();
		TestCase.assertEquals(51.0, addition.add(20.8, 30.2));
	}
}

public class SubtractionTest {

	@Test
	public void subtractIntTest() {
		Subtraction subtraction = new Subtraction();
		TestCase.assertEquals(-10, subtraction.subtract(20, 30));
	}
	@Test
	public void subtractDoubleTest() {
		Subtraction subtraction = new Subtraction();
		TestCase.assertEquals(-9.4,
				subtraction.subtract(20.8, 30.2));
	}
}
@RunWith(Suite.class)
@SuiteClasses({AdditionTest.class,SubtractionTest.class,
DivisionTest.class,Parameterized.class})
public class TestCases {
}
public class TestRunner {
	public static void main(String[] args) {
		Result result = JUnitCore.runClasses(AdditionTest.class);
		for (Failure failure : result.getFailures()) {
			System.out.println(failure.getMessage());
		}
		System.out.println(result.wasSuccessful());
	}
}
```

# JWebUnit

Extension of JUnit framework, JWebUnit is an API which is used to test Web Application. WebTester is a class which helps to test any web page. There are so many non-static methods in this class, some of them are as follows.

- getTestContext() : TestContext
- beginAt(String) : void
- assert***(String) : void
- click***(String) : void
- submit() : void
- etc…

```
public class Tester extends TestCase {
	private WebTester tester = new WebTester();

	@Override
	protected void setUp() throws Exception {
		tester.getTestContext().setBaseUrl(BASE_URL);
	}

	@Test
	public void testMethod() {
		tester.beginAt(PAGE_URL);
		tester.assertLinkPresentWithText(TEXT_OF_HIPERLINK);
		tester.clickLinkWithText(TEXT_OF_HIPERLINK);
	}
}
```