**@Test**

Method should be **public void**

**public void testSomeMethod()**

The **Test** annotation supports two optional parameters. The first, expected, declares that a test method should throw an exception. If it doesn't throw an exception or if it throws a different exception than the one declared, the test fails. For example, the following test succeeds:

@Test(**expected=IndexOutOfBoundsException.class**) public void outOfBounds() {

new ArrayList<Object>().get(1);

}

The second optional parameter, timeout, causes a test to fail if it takes longer than a specified amount of clock time (measured in milliseconds). The following test fails:

@Test(**timeout=100**) public void infinity() {

while(true);

}

**@Before :** When writing tests, it is common to find that several tests need similar objects created before they can run. Annotating a public void method with @Before causes that method **to be run before the** [**Test**](http://junit.org/junit4/javadoc/latest/org/junit/Test.html) **method**. **The @Before methods of superclasses will be run before those of the current class,** unless they are overridden in the current class. No other ordering is defined.

**@After :** If you allocate external resources in a [org.junit.Before](eclipse-javadoc:%E2%98%82=neptune-integration-service-app/C:%5C/Users%5C/IBM_ADMIN%5C/.m2%5C/repository%5C/junit%5C/junit%5C/4.12%5C/junit-4.12.jar%3Corg.junit(After.class%E2%98%83After%E2%98%82org.junit.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 [org.junit.Test](eclipse-javadoc:%E2%98%82=neptune-integration-service-app/C:%5C/Users%5C/IBM_ADMIN%5C/.m2%5C/repository%5C/junit%5C/junit%5C/4.12%5C/junit-4.12.jar%3Corg.junit(After.class%E2%98%83After%E2%98%82org.junit.Test) method. **All @After methods are guaranteed to run even if a** [**org.junit.Before**](eclipse-javadoc:%E2%98%82=neptune-integration-service-app/C:%5C/Users%5C/IBM_ADMIN%5C/.m2%5C/repository%5C/junit%5C/junit%5C/4.12%5C/junit-4.12.jar%3Corg.junit(After.class%E2%98%83After%E2%98%82org.junit.Before) **or** [**org.junit.Test**](eclipse-javadoc:%E2%98%82=neptune-integration-service-app/C:%5C/Users%5C/IBM_ADMIN%5C/.m2%5C/repository%5C/junit%5C/junit%5C/4.12%5C/junit-4.12.jar%3Corg.junit(After.class%E2%98%83After%E2%98%82org.junit.Test) **method throws an exception.** **The @After methods declared in superclasses will be run after those of the current class, unless they are overridden in the current class.**

**@BeforeClass :** Sometimes several tests need to share computationally expensive setup (like logging into a database). While this can compromise the independence of tests, sometimes it is a necessary optimization. **Annotating a public static void no-arg method with @BeforeClass** causes it to be **run once before any of the test methods in the class**. The @**BeforeClass methods of superclasses will be run before** those of the current class, unless they are shadowed in the current class.

**@AfterClass :** If you allocate expensive external resources in a [BeforeClass](http://junit.org/junit4/javadoc/latest/org/junit/BeforeClass.html) method you need to release them after all the tests in the class have run. **Annotating a public static void method with @AfterClass causes that method to be run after all the tests in the class have been run**. **All @AfterClass methods are guaranteed to run even if a** [**BeforeClass**](http://junit.org/junit4/javadoc/latest/org/junit/BeforeClass.html) **method throws an exception**. **The @AfterClass methods declared in superclasses will be run after those of the current class**, unless they are shadowed in the current class.

**@RunWith(some.class) :** When a class is annotated with @RunWith or extends a class annotated with @RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit. We added this feature late in development. While it seems powerful we expect the runner API to change as we learn how people really use it. Some of the classes that are currently internal will likely be refined and become public.

Test Suite

@RunWith(Suite.**class**)

@SuiteClasses({ ArraysCompareTest.**class**, QuickBeforeAfterTest.**class**, StringHelperParameterizedTest.**class**,

StringHelperTest.**class** })

**public** **class** AllTests {

}

assertArrayEquals : test the two array elements are same

**Mockito**

@RunWith(**MockitoJUnitRunner**.**class**)

**public** **class TestClass**

@Mock

ToDoService service; //Mock object like this or

@Test

**public** **void** testToDos() {

ToDoService service = Mockito.*mock*(ToDoService.**class**);//or we can mock like this.

*when*(service.someMethod()).thenReturn(Mockito.*anyInt*());

ToDoBusinessImpl impl = **new** ToDoBusinessImpl(service);

Assert.*assertEquals*(Mockito.*anyInt*(),impl.getTodos());

}