**Junit**

@Test(timeout=100)

@Test (expected = Exception.class)

The annotations @BeforeClass and @Before are same in functionality. The only difference is the method annotated with @BeforeClass will be called **once per test class** based, and the method annotated with @**Before** will be **called once per test** based.

**@After**

@After methods are **guaranteed** to run even if a Before or Test method throws an exception. The @After methods declared in superclasses will be run after those of the current class.

@**Ignore**: to temporarily disable a test or a group of tests.

**assertNotSame**(msnt.getPropValue("key1"), msnt.getPropValue("key2"));

**mockitto:**

<http://www.vogella.com/tutorials/Mockito/article.html>

**public** **class** **MockitoTest** {

**@Mock**

MyDatabase databaseMock;

@Rule **public** MockitoRule mockitoRule = **MockitoJUnit.rule();**

@Test

**public** **void** testQuery() {

ClassToTest t = **new ClassToTest(databaseMock**);

**boolean** check = **t.query("\* from t");**

assertTrue(check);

**verify(databaseMock).query("\* from t");**

}

}

@Spy or the spy() method can be used to wrap a real object.

when(spy.get(0)).thenReturn("foo");

We can use the verify() method on the mock object to verify that the specified conditions are met

@InjectMocks annotation which tries to do constructor, method or field dependency injection based on the type