**Lesson 14: Introduction to Junit 4**

**&**

**Lesson 18: Advanced Testing**

**1.Explain @Test with all attributes like timeout, expected**

**@Test:**

It is used to signify a method as a test method. There are some options which can be mentioned with this annotation.

**@Test** **with timeout**:

@Test(timeout=100) fails if the test takes longer than 100 milliseconds for

execution. This can be used to test infinite loops. Earlier you would have to

start every method with ‘test’ which is now not required. The method can be

named whatsoever. You have to simply prefix it with @Test annotation.

**@Test with expected:**

It is ideally to check that exceptions are thrown correctly by methods. Use the expected parameter in @Test annotation to test the exception that should be thrown.

**Eg:** @Test (expected=ArthemeticException.class)

         Public void divideByZeroTest()

{

            Calobj.divide(15,0);

             }

**2. Explain @ignore:**

**@ignore:**

* It will ignore the test method. This can be useful when the base code has been changed but the test is yet to be revised. You can also use this annotation when you do not want to run a test currently because it takes too long. This annotation should be used either in before or after the @Test Annotation.
* In addition to this, even a class can be annotated with @Ignore, and all the tests in that class will be ignored.
* The @Ignore annotation is should ideally be used when the method cannot be tested in some form and it is documented in the code. This should be a special case and warrant a discussion or code review to see if there is any way to test it.

**3. Explain static import of assert class.**

**Import static.org.junit.Assert.:**

JUnit4.x, the assert methods are static. A set of assertion methods useful for writing tests. Hence, you need to do either of the following:

1. Call assert methods using Assert.assertEquals().
2. While writing the test methods using static import we need to just specify @Test Annotation at starting of the method.

**3. Explain-@RunWith (Suite.class), @Suite.SuiteClasses.**

**@RunWith(Suite.Class)**

The @RunWith is the annotation which tells the JUnit runner to use the org.junit.runner.Suite class for running a particular class.

**@Suite.(Suite.Class)**

The @Suite annotation instructs the suite runner about the test classes which have to be included in this suite and the sequence in which they should be introduced.

**4. Explain about @Before, @After, @BeforeClass, @AfterClass**

**@Before**

Executed before each test. It is used to prepare the test environment (e.g., read input data, initialize the class).

**@After**

Executed after each test. It is used to clean up the test environment (e.g., delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.

**@BeforeClass**

Executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as static to work with JUnit.

**@AfterClass**

Executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as static to work with JUnit.

**6. What is parameterized Test?**

**Parameterized Test Cases:-**

JUnit 4 has introduced a new feature called parameterized tests. Parameterized tests allow a developer to run the same test over and over again using different values. There are five steps that you need to follow to create a parameterized test.

* Annotate test class with @RunWith (Parameterized.class).
* Create a public static method annotated with @Parameters that returns a Collection of Objects (as Array) as test data set.
* Create a public constructor that takes in what is equivalent to one "row" of test data.
* Create an instance variable for each "column" of test data.
* Create your test case(s) using the instance variables as the source of the test data.

The test case will be invoked once for each row of data.