jUnit

jUnit

- www.junit.org (https://github.com/junit-team/junit/wiki)
- "JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks."

- Assertions
- Annotations
- Test Suites
- Test Execution Order
- Testing Exceptions
- assertThat()
- Ignoring Tests
- Timeout for Tests
- Parameterized Tests
- Assumptions
- Rules

“Hello World” Test Case

```
@Test
public void testHelloWorld() {
	fail("World not yet implemented");
}
```

```
@Test
public void testHelloWorld() {
	assertEquals("Hello", "Hello");
}
```

Expected value

Actual value

Assertions

Assertion	Description
assertArrayEquals	Asserts that two arrays are equal
assertEquals	Asserts that two objects are equal
assertTrue	Asserts that a condition is true
assertFalse	Asserts that a condition is false
assertNull	Asserts that an object is null
assertNotNull	Asserts that an object is not null
assertSame	Asserts that two objects refer to the same object
assertNotSame	Asserts that two objects do not refer to the same object

Annotations

- @Test
- @Before and @After
- @BeforeClass and @AfterClass

```
@BeforeClass
public static void setUpBeforeClass() throws Exception {
    Database.connect();
}

@AfterClass
public static void tearDownAfterClass() throws Exception {
    Database.disconnect();
}
```

Test Suites

- Manually build a suite containing tests from many classes

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
  TestFeatureLogin.class,
  TestFeatureLogout.class,
  TestFeatureNavigate.class,
  TestFeatureUpdate.class
})
```

Test Execution Order

- Avoid writing tests which are dependent on other tests!
- But, if there is a needed to enforce order of test execution:

```
import org.junit.FixMethodOrder;
import org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class Tests {

}
```

- Again, try to find and eliminate dependency in your tests!

Testing Exceptions (example 1)

- We need to have a test case which will pass when an exception is thrown, to verify this behaviour as well
- There are three possibilities to do that

- 1. **expected** parameter

```
@Test(expected=IndexOutOfBoundsException.class)
 public void testEmptyArray() {
      new ArrayList<Object>().get(0);
 }
```

Testing Exceptions (example 2)

- 2. **try/catch** combination

```
@Test
public void testEmptyArray() {
  try {
    new ArrayList<Object>().get(0);
    fail("Expected an Exception to be thrown");
  } catch (IndexOutOfBoundsException e) {
    assertEquals(e.getMessage(), "Index: 0, Size: 0");
  }
}
```

Testing Exceptions (example 3)

- 3. Using ExpectedException **@Rule**

```
@Rule
public ExpectedException ex = ExpectedException.none();

@Test
public void testEmptyArray() throws IndexOutOfBoundsException
{
      ex.expect(IndexOutOfBoundsException.class);
      ex.expectMessage("Index: 0, Size: 0");
      new ArrayList<Object>().get(0);
  }
```

assertThat()

- Usage of Matchers from Hamcrest framework

```
public class TestMatcher {
    @Test
    public void testAssertThat() {
        assertThat(15, is(greaterThan(20)));
    }

    @Test
    public void testAssertTrue() {
        assertTrue(15>20);
    }
}
```

assertThat() – Output result

assertThat(15, is(greaterThan(20)));

java.lang.AssertionError:
Expected: is a value greater than <20>
but: <15> was less than <20>

assertTrue(15>20);

java.lang.AssertionError

Ignoring Tests

- Simply remove @Test
- More appropriate way is the usage of @Ignore

```
@Ignore("Test is ignored for now...")
@Test
public void testMath() {
   assertThat(15, is(not(greaterThan(20))));
}
```

Timeout for Tests

- Tests that run for ”too long” could be automatically declared as failed
- Two ways to achieve this:
  - **timeout** parameter in @Test annotation for a particular test

```
@Test(timeout=1000)
public void testWithTimeout() throws InterruptedException {
   Thread.sleep(2000);
}
```

  - Timeout @Rule (applies for every test in class!)

```
@Rule public Timeout globalTimeout = new Timeout(1000);
```

Parameterized Tests

- Replacement for writing several tests where the only difference is in the input and expected values

```
@RunWith(Parameterized.class)
public class ParameterizedTest {

   @Parameters(name = "TC#{index} with value={0} ")
   public static Collection<Object[]> data() {
      Object[][] data = new Object[][] { {1}, {2}, {3}, {4} };
      return Arrays.asList(data);
   }

   private int input;

   public ParameterizedTest(int input) {
      this.input= input;
   }

   @Test
   public void test() {
       assertThat(3, is(greaterThan(input)));
   }
}
```

Assumptions

- Possibility to run tests conditionally
- Some resources may not be available on developer machine, but they are on the integration server

```
@Test
public void testLogin() {
    assumeThat(Database.connect(), is(notNull()));
    // assertThat(…)
 }
```