JUnit Notasyonları Nelerdir ?

JUnit en çok kullanılan java test framework'üdür. Testlerimizi Java sınıfı şeklinde yazıp çalıştırırız. JUnit her test sınıfını belirli aşamalarla test eder. Test başlar başlamaz ilk önce @BeforeClass annotation'u çağrılır. Bu method her sınıf için sadece bir kere çağrılır. @Before ve @After ise her @Test methodundan önce ve sonra çağrılır. Tüm methodlar bitip test sınıfı sona ermeden önce ise @AfterClass annotation'u çağrılır. Bu doğrultuda JUnit notasyonlarını;

1. @BeforeClass
2. @AfterClass
3. @Before
4. @After
5. @Test
6. @Ignore

Olmak üzere altıya ayırabiliriz.

@BeforeClass

Bütün testler başlamadan önce çalışacak metottur. Örneğin, veritabanı bağlantısı bu sınıfta gerçekleştirilir.

@AfterClass

Bütün testler bittikten sonra çalışacak metottur. Yukarıda bahsetmiş olduğumuz veritabanı bağlantı işlemi @BeforeClass notasyonu ile başlatılırken veritabanı bağlantısını sonlandırma işlemi ise burada gerçekleştirilmektedir.

@Before

Before notasyonu da test adımlarından önce başlamaktadır. Veri okuması ve testin öncesinde yapılması gereken işlemler test adımlarını hızlandırmak için yani zamandan tasarruf etmek için bu metot da tanımlanır. Model ve nesnelerine atama işlemlerinin burada gerçekleştiririz.

@After

After notasyonu ise test metodlarından sonra çalışmaktadır. After notasyonun da en çok yapılan işlem, test metodlarından sonra her seferinde atanan değerlere null değerler gönderilmektedir.

- @Before ve @After notasyonları her test metodundan sonra çalışabilirken
- @BeforeClass ve @AfterClass bir kere çalışmaktadır.

@Test

Test notasyonu içerisinde test case'mizde yer alacak adımları tanımlarız.

Public methodlar için test oluşturacaksak bu annotationu kullanırız ama private methodlar için oluşturacaksak @org.junit.Test annotationunu ekleriz. @Test(timeout=500) eğer süreli test edeceksek ve testin exceptionlardan etkilenmesini istemiyorsak @Test(expected=IllegalArgumentException.class) bu annotationu kullanırız.

@Ignore

Yazdığımız test metotlarının çalıştırılmasını istemiyorsak bu amaç ile kullanabileceğimiz bir notasyondur. @Ignore notasyonunu ilgili test metodundan önce tanımlamamız yeterli olacaktır.

Örnek 1:

---------->>@Before - @Test - @After

```
    @Before

    public void setUp(){
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(15));
    }

    /**
    bir class da;
     -->@before ile olusturulan method class icindeki her test method'undan once ve 1 kez calisir ve
     genellikle public  void setup() seklinde yazilir
     -->@after ile olusturlan method class icindeki her test method'undan sonra ve 1 kez calisir ve
     genellikle public void tearDown() seklinde yazilir
     */

    @Test

    public void method1(){
        //Burada bir defa before methodu çalışır

        driver.get("https://amazon.com");

        //Bir defada after methodu çalışır.
    }

    @Test
    public void method2(){
        //Burada bir defa before methodu çalışır

        driver.get("https://www.hepsiburada.com/");

        //Bir defada after methodu çalışır.
    }

    @After
    //After notasyonu her testten sonra çalışır

    public void tearDown() { driver.close(); }
```

Örnek 2:

---------->>@BeforeClass - @Test - @AfterClass -@Ignore

```java
    /**
    before after da her test için farklı browser açıp kapatıyor
    beforeClass AfterClass'ta ise tek browser'dan testleri açıp kapatıyor
     */

    /**
    BeforeClass ve AfterClass notasyonları kullanıyorsak oluşturacağımız method'u static yapmamız gerekiyor
    */
    7 usages
    static WebDriver driver;

    @BeforeClass
    public static void setUp() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(15));
    }

    @Test
    @Ignore
    /**
    Çalışmasını istemediğimiz test için @Ignore notasyonu kullanılır
     */
    public void method1() throws InterruptedException {
        Thread.sleep( millis: 2000);
        driver.get("https://amazon.com");
    }
    @Test
    public void method2() throws InterruptedException {
        Thread.sleep( millis: 2000);
        driver.get("https://techproeducation.com");
    }
    @Test
    public void method3() throws InterruptedException {
        Thread.sleep( millis: 2000);
        driver.get("https://hepsiburada.com");
    }
    @AfterClass
    public static void tearDown() { driver.close(); }
}
```

JUnit Assertions:

1. JUnit Assertions Ne İçin Kullanılır ?

JUnit Assertions Kullanımı yazmış olduğumuz test senaryosunun sonunda elde edilen sonuç ile beklenilen sonuçların karşılaştırılması yapılması için kullanılır. Bir test senaryosundan beklenen sonuç ile elde edilmiş olan sonuç aynı ise test başarılı olarak nitelendirilir. Örneğin yazılan testler koşturma sırasında 3 test senaryosu geçti ve 2 test senaryosu doğru şekilde testten geçemediyse başarılı bir sonuç elde etmiş olmuyoruz. Assertions kullanımında birebir aynı sonuçların gelmesini beklememiz gerekmektedir.

2. JUnit Assertions Koşulları Nelerdir ?

Testlerde kabul edilen iki tür assertions vardır. Bunlardan biri Hard Assertions diğeri ise Soft Assertions'dır. Adından da anlaşılacağı gibi Hard Assertions ; test senaryosu içerisinde geçen onaylama koşulu gerçekleştirilmeden bir sonraki test adımına geçilmez. Bunun için bir otomasyonda bir exception attırılır. Exception atıldığında test durumu başarısız olarak sonuçlandırılır.Soft Assertions ise test senaryosu içerisinde bir onaylama koşulu gerçekleştirilmese bile test adımına devam eder ve herhangi bir exception atmaz ve bir sonraki test senaryosu adımı ile devam eder.

JUnit Assertions Kullanımı:

JUnit Assertions kullanımı geçmeden önce kaç çeşit assertions var bunu görelim. Sekiz adet assertions tipi vardır. Bunlar;

1. assertEquals
2. assertTrue
3. assertFalse
4. assertNull
5. assertNotNull
6. assertSame
7. assertNotSame
8. AssertArrayEquals

1. assertEquals

assertEquals , beklenen sonucu gerçek sonuç ile karşılaştırmak için kullanılır. Beklenen sonuç ile gerçek sonuç eşit değil ise gerçekleştirilen test senaryosu sonucunda assertionError hatası fırlatır.

2. assertTrue

assertTrue, beklenen bir sonucun true olduğunun kabul edilmesi gerektiği zaman kullanılır. Parametre olarak iki değer alır. İlk parametre de bir mesaj gönderilir ikinci parametrede ise gönderilen mesajın doğruluğu için koşul belirlenir.

3. assertFalse

assertFalse, beklenen bir sonucun false olması durumunda kullanılır. İki parametre alır. Parametrelerden biri mesajdır diğeri ise koşuldur. assertFalse ile koşul yerine getirilmez ise assertionError hatası fırlatır.

4.assertNull

assertNull, beklenen bir sonucun null olup olmadığı kontrol edilmesi için kullanılır. Bir nesneyi parametre olarak alır ve nesne null değil ise assertionError hatası fırlatır.

5.assertNotNull

assertNotNull, beklenen bir sonucun null olmadığını doğrulamak için kullanılır. Bir nesneyi parametre olarak alır ve nesne null ise assertionError hatası fırlatır.

6.assertSame

assertSame, parametre olarak verilen iki nesnenin aynı nesneye karşılık gelip gelmediğini kontrol eder. Eğer nesneler aynı nesneyi karşılamıyor ise assertionError hatası fırlatır.

7.assertNotSame

assertNotSame, parametre olarak verilen iki nesnenin birbirine eşit olmadığı kontrolünü eder. Eğer aynı nesneye karşılık geliyor ise assertionError hatası fırlatır.

8.assertArrayEquals

assertArrayEquals, parametre olarak verilen iki dizinin eşit olup olmadığını kontrol eder. Her iki dizi içinde null değeri var ise bunlar eşit olarak kabul edilir. Eğer eşit değil ise assertionError hatası fırlatır.

Örnek 3:

--------->> Assert

```
35      static WebDriver driver;
36      @BeforeClass
37      public static void setUp() {
38          WebDriverManager.chromedriver().setup();
39          driver = new ChromeDriver();
40          driver.manage().window().maximize();
41          driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(15));
42          driver.get("https://www.amazon.com");
43      }
44      @AfterClass
45      public static void tearDown() { driver.close(); }
48      @Test
49      public void test1(){
50          //a-Url'nin facebook içerdiğini test edelim
51          String expectedUrl = "facebook";
52          String actualUrl = driver.getCurrentUrl();
53          Assert.assertFalse(actualUrl.contains(expectedUrl));
54          //Assert.assertNotEquals(expectedUrl,actualUrl);
55      }
56      @Test
57      public void test2(){
58          //b-Title'in facebook içermediğini test edelim
59          String actualTitle = driver.getTitle();
60          String expectedTitle = "facebook";
61          Assert.assertFalse(actualTitle.contains(expectedTitle));
62      }
63      @Test
64      public void test3(){
65          //c- sol üst köşede amazon logosunun göründüğünü test edelim
66          WebElement logo = driver.findElement(By.id("nav-logo-sprites"));
67          Assert.assertTrue(logo.isDisplayed());
68      }
69      @Test
70      public void test4(){
71          //d- Url'nin www.facebook.com olduğunu test edin
72          String expectedUrl = "www.facebook.com";
73          String actualUrl = driver.getCurrentUrl();
74          Assert.assertNotEquals(expectedUrl,actualUrl);
75      }
76
```

Assert Özet:

Assert, bir test senaryosunun PASS veya FAILED durumunu belirlemede kullanilan yararlı bir yöntemdir. Secilen metot ve yazilan boolean kosul'a gore test sonucu belirlenir.

Assert yöntemleri, Java.lang.Object Class'ina bagli org.junit.Assert Class'i tarafından sağlanır.

En cok kullandigimiz 3 Assert metodu;

1) Assert.assertTrue(koşul)

Yazilan kosul'un sonucu True ise test PASS, yoksa test FAILED olur

Assert.assertTrue(20 > 15) → Test PASSED
(20 > 15 : True)

Assert.assertTrue(10 > 30) → Test FAILED
(10 > 30 : False)

2) Assert.assertFalse(koşul)

Yazilan kosul'un sonucu False ise test PASS, yoksa test FAILED olur

Assert.assertFalse(40 > 50) → Test PASSED
(40 > 50 : False)

Assert. assertFalse(30 > 20) → Test FAILED
(30 > 20 : True)

3) Assert.assertEqual(expected, actual)

Yazilan expected ile actual esit ise test PASS, yoksa test FAILED olur

Assert.assertEquals("Ali" , "Ali") → Test PASSED
("Ali" , "Ali" : True)

Assert. assertEquals(30 , 20) → Test FAILED
(30 , 20 : False)

Kaynak:

- https://www.mobilhanem.com/junit-notasyonlari/
- https://techproeducation.com/