Introducere

În continuare vom prezenta unul dintre instrumentele principale utilizate în testarea automatizată pentru a crea teste unitare sau a utiliza metode de analiză și comparare a rezultatelor obținute – biblioteca Junit.

Introducere Junit

Junit este un schelet (framework) popular destinat scrierii de teste unitare în ecosistemul Java. Acest instrument vine în diferite versiuni, iar una dintre cele mai populare este versiunea 5, care urmează a fi utilizată în cadrul cursului dat. Junit este folosit în mod prioritar pentru scrierea de teste unitare, care reprezintă responsabilitatea dezvoltatorilor. Cu alte cuvinte, un dezvoltator care scrie o funcție(metodă) în cadrul programului, este responsabil să scrie și un set de teste unitare care permite validare acestei metode. Știind rezultatul așteptat în urma executării unei metode predefinite, dezvoltatorul va apela metoda cu parametrii actuali prestabiliți și va compara rezultatul obținut cu cel așteptat. Spre exemplu, avem metoda divide(int a, int b) care face împărțirea numărului a la b. Un test unitar pentru această metodă ar verifica dacă divide(5,1) este egală cu 5. În mod respectiv ar putea fi verificat dacă divide(5,0) aruncă o excepție având în vedere că împărțirea la 0 nu este posibilă. Utilizarea scheletului (framerwork-ului) Junit atrage după sine un șir de avantaje precum:

- Îmbunătățirea vitezei de scriere a codului și creșterea calității acestuia;
- Clasele de test sunt ușor de scris și modificat pe măsură ce codul sursă se mărește, putând fi compilate împreună cu codul sursă al proiectului;
- Clasele de test JUnit pot fi rulate automat (în suită), rezultatele fiind vizibile imediat.
- Clasele de test măresc încrederea programatorului în codul sursă scris și îi permit să urmărească mai ușor cerințele de implementare ale proiectului;
- Metodele Junit permit scriere testelor pentru orice nivel, nu doar unitar (unit);
- Testele pot fi scrise înaintea implementării, fapt ce garantează înțelegerea funcționalității de către dezvoltator.

Având în vedere caracterul generic al posibilităților oferite de către Junit, în continuare se va folosi metodele Junit pentru a valida dacă un element web deține sau nu proprietățile așteptate. Pentru a putea însă purcede la această etapă este mai întâi necesar de-a configura librăria în cadrul propriului proiect de testare.

Configurarea Junit

Este necesar să menționăm că bibliotecile externe pot fi importate și utilizate cu ajutorul instrumentului Maven. Pentru a seta și configura Junit se va apela la ajutorul acestuia, mai exact se verifica repozitoriul al dependențelor Maven.

Așadar, odată cu actualizarea proiectului de tip Maven se va putea apela metodele din cadrul bibliotecii Junit care vor permite crearea scenariilor de testare complete, care includ și partea de validare. Pornind de la scopul în cadrul proiectelor de a crea scenarii de testare automatizate pentru testarea întregului sistem dar nu unitară.

Crearea unui Scenariu de testare cu Junit

Rularea unor teste utilizând librăria Junit este simplă și implică utilizarea Java. În cazul versiunii 5+ de Junit este necesar ca proiectul să fie configurat cu ajutorul Java 11+. Iar crearea unor teste are

loc într-o clasă separată, dedicată acestui scop. Pornind de la ideea că proiectul este deja creat, în cadrul cursului vom utiliza clasa TestRunner pentru a scriere teste Junit. Cu toate acestea, testele pot fi create și în clase noi create. Singura recomandare față de numele unei clase dedicată scrierii de teste este ca această să conțină în incinta numelui său cuvântul ”Test”.

Fiecare test este implementat în cadrul corpului unei metode. Adnotația @Test moștenită din cadrul librăriei Junit este cea care face distincția dintre o metodă obișnuită și una care reprezintă un test. Odată ce o clasă conține cel puțin un test, aceasta va putea fi rulată fără a conține o metodă main în ea. Prin urmare, dacă ne referim la testul scenariu scris, atunci implementarea lui arată astfel într-o metodă Junit:

```
public class TestRunner {

    @Test
    @DisplayName("Registration form is completed")
    public void fillInTheRegisterPage() throws InterruptedException {
                WebDriverManager webDriverManager
                = new WebDriverManager( driverType: "CHROME");
        webDriverManager.getDriver().get("https://demo.opencart.com/index.php?route=common/home");


        HomePage homePage = new HomePage(webDriverManager.getDriver());
        homePage.clickOnMyAccountBtn();
        homePage.clickOnRegisterButton();

        RegisterPage registerPage = new RegisterPage(webDriverManager.getDriver());
        registerPage.fillInTheRegistrationForm( firstName: "Emilia", lastName: "Cretu",  email: "Emilia56@gmail.com",
                phone: "079998577", password: "paswordws!s");
        registerPage.clickPrivacyCbx();
        registerPage.clickContinueButton();

        Thread.sleep( time: 10000);
        webDriverManager.closeDriver();
    }
}
```

Clasa de mai sus conține un singur test, al cărui nume este dictat în incinta adnotației @DisplayName astfel, în urma execuției vor fi afișate rezultatele testului în felul următor:

Execuția testului prevede executarea codului din cadrul corpului metodei. În mod similar pot fi create mai multe teste în cadrul unei clase, prin urmare următorul test va fi unul complet, adică va include și partea de validare iar drept scop va avea verificarea dacă utilizatorul poate ajunge pe pagina de înregistrare apăsând butoanele disponibile pe prima pagină.

Cel de-al doilea test conține și o instrucțiune nouă, o metodă statică numită assertTrue și apelată din cadrul clasei Assertions. Aceasta este cea care validează dacă testul a trecut sau nu. Fără a include instrucțiuni de validare precum asert – testele nu au sens, deoarece ele rezultatul obținut nu va fi comparat niciodată cu cel așteptat. În cazul de mai sus, odată ce se tastează butonul de înregistrare utilizatorul este direcționat către pagina de Înregistrare, al cărei url conține cuvântul ”register”. În cazul în care sistemul nu funcționează corespunzător iar pagina respectivă nu va fi deschisă, atunci testul va cădea iar inginerul QA va raporta un defect.

```java
@Test
@DisplayName("Registration page is accessible from home page")
public void registerPageIsDisplayed() throws InterruptedException {
    WebDriverManager webDriverManager
            = new WebDriverManager( driverType: "CHROME");
    webDriverManager.getDriver().get("https://demo.opencart.com/index.php?route=common/home");


    HomePage homePage = new HomePage(webDriverManager.getDriver());
    homePage.clickOnMyAccountBtn();
    homePage.clickOnRegisterButton();

    assertTrue(webDriverManager.getDriver().getCurrentUrl().contains("register"));

    webDriverManager.closeDriver();
}
```

Nu există un număr limitat de teste care pot fi scrise într-o clasă, cu toate acestea se recomandă excluderea codului redundat și utilizarea unor instrucțiuni de validare cât mai elocvente. În continuare se propune studierea modului prin care codul poate fi scris mai restrâns iar implementarea testelor mai ușoară. Una dintre căile care permit rezolvarea problemei date este utilizarea adnotațiilor puse la dispoziție de către dependențele Junit.

Utilizarea adnotațiilor BeforeEach și AfterEach

Junit definește un set de adnotații care indică ordinea de execuția a metodelor în momentul rulării. Dacă ne referim la exemplul de mai sus, în ambele teste este inclusă secțiunea de cod care creează un web driver și care îl închide. Având în vedere că absolut toate testele automatizate ale interfeței utilizatorului prevăd aceste măsuri de deschidere și închidere a browser-ului, se face inutil rescrierea codului în cadrul fiecărui test, deoarece Junit facilitează soluționarea problemei cu ajutorul adnotațiilor BeforeEach și AfterEach.

Adnotația BeforeEach este specificată la nivel convențional drept @BeforeEach. Prezența acesteia indică despre existența unei metode care va fi executată înainte oricărui test executat. Adesea, metoda definită cu adnotația @BeforeEach poartă numele de setUp() și include un set de instrucțiuni de cod menite să configureze mediul de testare înaintea fiecărui test:

```java
@BeforeEach
public void setUp(){
    webDriverManager = new WebDriverManager( driverType: "CHROME");
    webDriverManager.getDriver().get("https://demo.opencart.com/index.php?route=common/home");

}
```

Conform exemplului de mai sus, ori de câte ori un test Junit va fi rulat, mai întâi de toate va fi inițializat un obiect al clasei webDriverManager, care nemijlocit va inițializa driverul prin apelul funcției de getDriver.

În mod similar, există adnotația AfterEach care este menită să execute procedurile de finalizare ale unui test. În cazul abordat în exemplele anterioare, asta ar însemna să închidă instanța de browser deschisă de către driver. Metoda definită cu această adnotație nu are specificații particulare față de atribuirea numelui, însă adesea se întâlnește drept tearDown():

```
@AfterEach
public void tearDown() {
    webDriverManager.closeDriver();
}
```

Blocul de cod din metoda tearDown lucrează după principiul codului din blocul finally studiat în cadrul lecției cu excepții datorită adnotației @AfterEach. Indiferend de rezultatul unui test, acest bloc va fi executat, respectiv excepțiile sau erorile apărute în timpul execuției nu reprezintă un impediment.

Pe lângă aceste adnotații des întâlnite mai există și altele, cum ar fi:

- @BeforeAll – specificată pentru metoda care se execută la începutul rulării tuturor testelor. Metoda definită cu această adnotare va fi executată o singură dată.
- @AfterrAll – specificată pentru metoda care se execută odată cu finisarea executării tuturor testelor. La fel ca în cazul de mai sus, aceasta se execută o singură dată.
- @RepeatedTest(n) – repetă un test de n ori.
- Etc.

Important de menționat la această etapă este faptul că adnotațiile din Junit diferă în dependență de versiunea instrumentului. Cele studiate mai sus sunt reprezentative pentru versiunea 5 Junit, care mai este numită și versiunea Jupiter. Acestea sunt întâlnite și în versiunea veche, însă cu alt nume. În cele din urmă ele au sens similar și sunt menite construirii unor seturi de teste.

O clasă de teste Junit, reprezintă o suită de teste. Prin urmare, se recomandă ca o suită de teste să dispună de metode de pre-setarea a mediului de testare definite cu ajutorul adnotației @BeforeAll, @BeforeEach și desigur de finalizare a procedurilor de testare @AfterAll sau/și @AfterEach. Aplicarea acestor instrumente în cadrul suite de două teste deja create este prezentată mai jos.

Prin urmare codul a fost redus substanțial iar acest efect de reducere va fi mai simțitor odată ce suita de teste va fi completată cu alte teste, ce ar fi utilizat aceleași condiții repetitive. Junit oferă și alte adnotații în afară de cele studiate menite scrierii de teste unitare, cu toate acestea, cunoașterea celor de mai sus reprezintă baza creării de teste funcționale pentru validarea interfeței utilizatorului. După cum se poate observa în unul dintre teste lipsește instrucțiunea de validare, ceea ce îl face inutil. Astfel, în continuare se prezintă instrucțiunile de validare propuse de către biblioteca Junit.

```java
public class TestRunner {

    WebDriverManager webDriverManager;

    @BeforeEach
    public void setUp() {
        webDriverManager = new WebDriverManager( driverType: "CHROME");
        webDriverManager.getDriver().get("https://demo.opencart.com/index.php?route=common/home");
    }

    @AfterEach
    public void tearDown() {
        webDriverManager.closeDriver();
    }

    @Test
    @DisplayName("Registration page is accessible from home page")
    public void registerPageIsDisplayed() throws InterruptedException {

        HomePage homePage = new HomePage(webDriverManager.getDriver());
        homePage.clickOnMyAccountBtn();
        homePage.clickOnRegisterButton();
        assertTrue(webDriverManager.getDriver().getCurrentUrl().contains("regihgster"));
    }

    @Test
    @DisplayName("Registration form is completed")
    public void fillInTheRegisterPage() throws InterruptedException {

        HomePage homePage = new HomePage(webDriverManager.getDriver());
        homePage.clickOnMyAccountBtn();
        homePage.clickOnRegisterButton();

        RegisterPage registerPage = new RegisterPage(webDriverManager.getDriver());
        registerPage.fillInTheRegistrationForm( firstName: "Emilia",  lastName: "Cretu",  email: "Emilia56@gmail.com",
                phone: "079998577",  password: "paswordws!s");
        registerPage.clickPrivacyCbx();
        registerPage.clickContinueButton();
        Thread.sleep( time: 10000);
    }
}
```

Instrucțiunile de validare – Asserts

Assert	Exemplu
assertEquals	assertEquals(4, calculator.multiply(2, 2),"mesaj de eroare opțional");
assertTrue	assertTrue('a' < 'b', () → " mesaj de eroare opțional ");
assertFalse	assertFalse('a' > 'b', () → " mesaj de eroare opțional ");
assertNotNull	assertNotNull(yourObject, " mesaj de eroare opțional ");
assertNull	assertNull(yourObject, " mesaj de eroare opțional ");

Prin instrucțiunile de validare se subînțelege o serie de metode statice care permit compararea a două entități. Aceste instrucțiuni sunt numite assert-uri și sunt disponibile în clasa

Asserts. În Junit assert-urile reprezintă un șir finit de clase de tip statice cu modificatorul public destinate comparării rezultatului obținut cu cel așteptat. Sub formă tabelară ele sunt afișate astfel:

În tabelul de mai sus sunt indicate cele mai des utilizate instrucțiuni de validare care urmează a fi utilizate în cadrul proiectului de testare. Însoțitor au fost prezentate și un set de exemple cum aceste instrucțiuni pot fi folosite. Primordial se observă utilizarea unui mesaj de eroare opțional, prezent în toate cazurile. Acesta este menit să ofere mai multă claritate din moment ce instrucțiunea de validare nu este executată cu succes. În două dintre exemple se utilizează proceduri de lambda din cadrul versiunii 8 Java, care nu au fost studiate, cu toate acestea, utilizarea lor este simplă și ușoară în acest context. Toate metodele (instrucțiunile) de validare de mai sus sunt supraîncărcate astfel încât să accepte diverse forme de parametri formali. Implementarea acestora poate fi găsită în clasa: `org.junit.jupiter.api.Assertions.*`.