TESTNG

TESTNG

- TestNG je testing framework koji počiva na JUnit-u i NUnit-u
- Dizajniran je tako da pokriva sve nivoe testiranja od jediničnog, funkcionalnog, do e2e testiranja

INSTALLING

- Novije verzije IntelliJ-a dolaze sa unapred ugrađenim TestNG framework-om
- Instalacija za Eclipse https://www.lambdatest.com/blog/how-to-install-testng-in-eclipse-step-by-step-guide/

Dependency koji je potrebno uključiti u pom.xml file Maven projekta

https://mvnrepository.com/artifact/org.testng/testng/7.6.1

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.testng/testng -->
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.6.1</version>
    <scope>test</scope>
</dependency>
```

XML SUITE

- XML je konfiguracioni fajl za TestNG. Koristi se za definisanje test suite-ova i test slučajeva
- Takođe se koristi i za prosleđivanje parametara metodama testiranja
- Testng.xml pruža različite opcije za uključivanje i isključivanje grupa testova, klasa, kao i pojedinačnih test metode u test suite. Pored toga, omogućava i konfiguraciju i objedinjavanje više testova iz različitih test klasa i njihovo pokretanje u višenitnom okruženju
- XML fajl je definisan korenim suite elementom, čiji naziv jedini obavezan atribut
- Suite može da sadrži jedan ili više test slučaj koji je definisan test elementom. Svaki test slučaj može da sadrži:
  - Test klase, koje je definisana putanjom do same klase (naziv paketa + naziv test klase). Test klase mogu opciono da sadrže test metode
  - Test pakete

suite-testng.xml

```
1  <suite name="Combine Suite" verbose="1">
2      <test name="Combine Test">
3          <packages>
4              <package name="firstpackage" />
5          </packages>
6          <classes>
7              <class name="secondpackage.FirstTestClass" />
8              <class name="thirdpackage.FirstTestClass">
9                  <methods>
10                     <include name="firstTest" />
11                 </methods>
12             </class>
13         </classes>
14     </test>
15 </suite>
```

INCLUDING AND EXCLUDING

- TestNG pruža fleksibilnost uključivanja ili isključivanja testova prilikom definisanja testnog paketa
- Ovo pomaže u definisanju test suita sa određenim skupom testova
- Uključivanje i isključivanje se može definisati na nivou:
  - Paketa
  - Klasa
  - Metoda
- Tom prilikom mogće je navoditi njihova puna imena, ali isto tako i koristiti regex izraze

regexp-testng.xml

```xml
<suite name="Regular Exp Suite" verbose="1">
    <test name="Regular Exp Test">
        <classes>
            <class name="regularexppackage.RegularExpClass">
                <methods>
                    <include name=".*Test.*" />
                </methods>
            </class>
            <class name="firstpackage.FirstTestClass">
                <methods>
                    <exclude name="firstTest" />
                </methods>
            </class>
        </classes>
    </test>
</suite>
```

BEFORE AND AFTER ANNOTATIONS

- Before i After anotacije se uglavnom koriste za izvršavanje određenog skupa koda pre ili posle izvršenja test metoda. Oni se u osnovi koriste za podešavanje sistemskih promenljivih ili konfiguraciju pre početka izvršavanja testova, a zatim i za čišćenje bilo koje od ovih stvari nakon završetka izvršenja testa.
- TestNG pruža pet različitih vrsta Before i After anotacija, od kojih svaka može se koristiti u zavisnosti od zahteva testa
  - `@BeforeSuite`
  - `@AfterSuite`
  - `@BeforeTest`
  - `@AfterTest`
  - `@BeforeGroups`
  - `@AfterGroups`
  - `@BeforeClass`
  - `@AfterClass`
  - `@BeforeMethod`
  - `@AfterMethod`

TEST ANNOTATIONS

- Jedna od osnovnih anotacija u svakom test frameworku je `Test` anotacija
- Ova anotacija označava metod ili klasu kao deo TestNG testa. Ako se primeni na nivou klase, deklarisaće da su sve javne metode te klase zapravo test metode
- Različite funkcionalnosti se dobijaju dodavanje atributa samoj anotaciji:
  - `alwaysRun`
  - `dataProvider`
  - `dataProviderClass`
  - `dependsOnGroups`
  - `dependsOnMethods`
  - `description`
  - `enabled`
  - `expectedExceptions`
  - `groups`
  - `timeout`
  - `Priority`

TestClass.java

```java
package test;

import org.testng.annotations.Test;

@Test
public class TestClass {

    public void testMethodOne(){
        System.out.println("Test method one.");
    }

    public void testMethodTwo(){
        System.out.println("Test method two.");
    }

    private void testMethodThree(){
        System.out.println("Test method three.");
    }
}
```

DISABLING A TEST

- Postoje scenariji u kojima je potrebno da se onemogući izvršavanje određenog test ili skupa testova
- Na primer, primećen je bug u funkionalnosti koda, što dovodi do toga da se testovi u određenim scenarijima ne mogu izvršiti. Pošto je problem već identifikovan, bolje je onemogućiti pomenute testne scenarije dok se bug ne reši
- Onemogućavanje testa se može postići u TestNG podešavanjem atributa `enable` za Test anotacije na false
- Ovo će onemogućiti da se navedeni metod testiranja izvrši kao deo testnog paketa
- Ako je ovaj atribut postavljen na nivou klase, sva javne metode unutar klase će biti onemogućene

DisableTestClass.java

```
1  package test;
2
3  import org.testng.annotations.Test;
4  public class DisableTestClass {
5
6      @Test(enabled=true)
7      public void testMethodOne(){
8          System.out.println("Test method one.");
9      }
10
11     @Test(enabled=false)
12     public void testMethodTwo(){
13         System.out.println("Test method two.");
14     }
15
16     @Test
17     public void testMethodThree(){
18         System.out.println("Test method three.");
19     }
20 }
```

exception/ExceptionTest.java
exception/ExceptionMessageTest.java

EXCEPTION TEST

- Tokom pisanja jediničnih testova mogu postojati određeni scenariji u kojima moramo da proverimo da li program izbacuje izuzetak tokom izvršavanja.
- TestNG pruža funkciju za testiranje takvih scenaria dozvoljavajući korisniku da odredi tip izuzetaka koji se očekuju da se baci u metodi koja se testira
- Ukoliko metoda koja se testira ne baci izuzetak ili baci izuzetak koji se ne očekuje, test će pasti
- Pored samih izuzetaka, moguće je očekivati i određenu poruku

ExceptionTest.java

```java
package exception;

import java.io.IOException;
import org.testng.annotations.Test;
public class ExceptionTest {

    @Test(expectedExceptions={IOException.class})
    public void exceptionTestOne() throws Exception{
        throw new IOException();
    }

    @Test(expectedExceptions={IOException.class,
            NullPointerException.class})
    public void exceptionTestTwo() throws Exception{
        throw new Exception();
    }
}
```

TIME TEST

- Tokom izvođenja testova može doći do slučajeva da se određeni testovi zaglave ili da mogu potrajati mnogo više vreme od očekivanog. U tom slučaju test treba da se proglasi neuspešnim.
- TestNG omogućava korisniku da konfiguriše vremenski period u kom treba da sačeka da se test u potpunosti završi. Ovo se može konfigurisati na dva načina:
  - Na nivou paketa (suite-a): biće primenljivo na sve testove u pomenutom TestNG test paketu
  - Na nivou test metoda: biće primenljivo za pomenutu metodu i ukoliko postoji, pregaziće vreme definisano na nivou suite-a

TimeMethodTest.java

```
1  package time;
2
3  import org.testng.annotations.Test;
4
5  public class TimeMethodTest {
6
7      @Test(timeOut=500)
8      public void timeTestOne() throws InterruptedException{
9          Thread.sleep( millis: 1000);
10         System.out.println("Time test method one");
11     }
12
13     @Test
14     public void timeTestTwo() throws InterruptedException{
15         Thread.sleep( millis: 400);
16         System.out.println("Time test method two");
17     }
18 }
```

PARAMETERIZATION OF TEST

- Jedna od važnih karakteristika TestNG-a je parametrizacija. Ova funkcija omogućava korisniku da prosledi vrednosti parametara za metode testiranja kao argumente.
- Postoje dva načina na koja možemo da obezbedimo vrednosti parametara za metode testiranja:
  - Kroz testNG.xml konfiguracionu datoteku
  - Putem DataProvider-a

PARAMETERIZATION THROUGH TESTNG.XML

- Koristi se kada je test metode potrebno parametrizovati jednostavnim podacima
- Podaci se definišu u testng.xml suite-u u sklopu parameter elementa, naziv atributom se definiše naziv parametra, dok se njegova vrednost smešta u okviru value atributa
- Test metode se dodatno anotiraju `Parameters` anotacijom, u kojoj se u nastavku navode isti nazivi parametara, kao u xml fajlu
- Parameters anotacija može biti korišćena i sa metodama anotiranim sa Before/After

ParameterTest.java

```java
package parameter;

import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

3 usages
public class ParameterTest {
    /**
     * Following method takes one parameter as input. Value of the
     * said parameter is defined at suite level.
     */
    @Parameters({ "suite-param" })
    @Test
    public void prameterTestOne(String param) {
        System.out.println("Test one suite param is: " + param);
    }
```

param-testng.xml

```xml
<suite name="Parameter test Suite" verbose="1">
    <parameter name="suite-param" value="suite level
 parameter" />
    <test name="Parameter Test one">
        <classes>
            <class name="parameter.ParameterTest">
                <methods>
                    <include name="prameterTestOne" />
                </methods>
            </class>
        </classes>
    </test>
    <test name="Parameter Test two">
```

PARAMETERIZATION WITH DATAPROVIDER

- Jedna od važnih karakteristika koje pruža TestNG je funkcionalnost DataProvider-a koja omogućava da korisnik piše testove zasnovane na podacima, što znači da se isti metod testiranja može pokrenuti više puta sa različitim skupovima podataka
- DataProvider pomaže u obezbeđivanju složenih parametara metodama testiranja kakve nije moguće proslediti iz XML-a
- Da bi se koristila funkcija DataProvider-a potrebno je deklarisati metod `DataProvider` anotacijom, a zatim koristite pomenuti metod u test metodi koristeći dataProvider atribut
- Po pravilu, DataProvider metoda kao povratnu vrednost mora da vrati niz nizova objekata

DataProviderInSameClassTest.java

```
1  package dataprovider;
2
3  import org.testng.annotations.DataProvider;
4  import org.testng.annotations.Test;
5
6  public class DataProviderInSameClassTest {
       1 usage
7      @DataProvider(name = "data-provider")
8      public Object[][] dataProviderMethod() {
9          return new Object[][] { { "data one" }, { "data two" } };
10     }
       1 usage
11     @Test(dataProvider = "data-provider")
12     public void testMethod(String data) {
13         System.out.println("Data is: " + data);
14     }
15 }
```

TEST GROUP

- TestNG grupe omogućavaju da se izvrši grupisanje različitih metoda testiranja. Grupisanje testnih metoda je korisno kada postoji potreba da pristupite test metodama različitih klasa
- TestNG-a nudi mogućnost izvršavanja samo određenog skupa grupa, dok se ostale grupe isključuju. Na ovaj način je moguće optimizovati broj testova koji je potrebno da se izvrši u datom trenutku
- Grupe se navode u XML fajlu `<groups>` elementom
- Grupe se mogu navesti u elementima `<suite>` ili `<test>`
  - Ako je element `<groups>` naveden unutar `<suite>` elementa, onda se pravila za uključivanje grupa primenjuju na sve test slučajeve XML Suite-a
  - Ako je element `<groups>` naveden u određenom `<test>` elementu, onda se primenjuje samo na testove iz tog test slučaja
- Test metodi se može dodeliti više grupa, koje se u tom slučaju odbajaju zarezima
- Prilikom pokretanje, xml fajlom moguće je uključiti ili isključiti grupe iz izvršavanja

TestGroup.java

```java
package group;

import org.testng.annotations.Test;
public class TestGroup {
    @Test(groups={"test-group"})
    public void testMethodOne(){
        System.out.println("Test method one belonging to group.");
    }

    @Test
    public void testMethodTwo(){
        System.out.println("Test method two not belonging to group.");
    }

    @Test(groups={"test-group"})
    public void testMethodThree(){
        System.out.println("Test method three belonging to group.");
    }
}
```

group-testng.xml

```xml
<suite name="Time test Suite" verbose="1">
    <test name="Timed Test">
        <groups>
            <run>
                <include name="test-group" />
            </run>
        </groups>
        <classes>
            <class name="group.TestGroup" />
        </classes>
    </test>
</suite>
```

DEPENDENCIES

- Zavisnost je karakteristika u TestNG-u koja omogućava da test metoda zavisi od pojedinačne ili grupe test metoda. Ovo će pomoći u definisanju skupa testova koji će se izvršiti pre same test metode
- Zavisnost između metoda se može ostvariti samo između test metoda koje se nalaze u istoj klasi ili ukoliko između klasa kojima pripadaju postoji nasleđivanje
- Test metoda koja je zavisna od druge test metode ili grupe test metoda će se izvršiti samo u slučaju da su uspešno izvršene sve test metode od kojih ona zavisi, u suprotnom biće preskočena
- Ukoliko je potrebno omogućiti da test metoda zavisi od testova iz različitih klasa, to se može postići tako što će se sve metode od kojih ciljana metoda zavisi dodeliti istoj grupi, a zatim će se test metoda posaviti da zavisi baš od te definisane grupe

DependencyTest.java

```java
package dependency;

import org.testng.annotations.Test;
public class DependencyTest {

    @Test(dependsOnMethods={"testTwo","testThree"})
    public void testOne(){
        System.out.println("Test method one");
    }
    1 usage
    @Test
    public void testTwo(){
        System.out.println("Test method two");
    }
    1 usage
    @Test
    public void testThree(){
        System.out.println("Test method three");
    }
}
```

TEST RUN ORDER

- Podrazumevano, TestNG testne klase i metode izvršava u alfabetskom poretku na osnovu njihovog naziva
- Međutim, korisnik ima mogućnost da utiče na redosled izvršavanja test metoda dodeljujući im prioritet izvršavanja
- Prioritet se može promeniti dodeljući celobrojnu vrednost `priority` atributu `@Test` anotacije. Najveći prioritet je 0 i smanjuje se porastom brojne vrednosti
- Brojčana vrednost `priority` atributa kreće se u opsegu od -5000 do 5000

priority/PriorityTest.java

```java
public class PriorityTest {

    @Test
    public void c_method() { System.out.println("I'm in method C"); }

    @Test(priority=1)
    public void b_method() { System.out.println("I'm in method B"); }

    @Test(priority = 100)
    public void a_method() { System.out.println("I'm in method A"); }

    @Test
    public void e_method() { System.out.println("I'm in method E"); }

    @Test
    public void d_method() { System.out.println("I'm in method D"); }
}
```

ASSERTIONS

- Asertacije predstavljaju ključnu funkcionalnost svakog testing frameworka kojim proveravamo rezultate testa
- Tvrdnje u TestNG-u su način da se proveri da li se očekivani rezultat i stvarni rezultat podudaraju ili ne
- Ukoliko ne dođe do poklapanja, test metoda baca `AssertionError`
- Test se smatra uspešnim ako je završen bez izbacivanja izuzetka ili ako je izbacio očekivani izuzetak
- Osnovni oblik asertacije

  `Assert.Method( actual, expected)`

  - actual - stvarna vrednost koja je dobijena pozivom metode koja se testira
  - expected - očekivana vrednost

```java
public class AssertEqualsTest {

    @Test
    public void testConcatenate() {
        ExampleUnit myUnit = new ExampleUnit();

        String result = myUnit.concatenate("one", "two");

        assertEquals( actual: "onetwo", result);
    }
}
```

LOGGING AND REPORTING

- Izveštavanje je jedan najvažnijih delova testiranja, jer pomaže korisnicima da razumeju rezultate izvršenja testova, tačku prekida i razloge neuspešnog izvršavanja testova
- TestNG podrazumevano generiše različite tipove izveštaja, tj. XML i HTML izveštaje. TestNG takođe omogućava pisanje sopstvenog report sistema korišćenjem
  - Listener-a
    - Za implementaciju klase Listener-a potrebno je implementirati `org.testng.ITestListener` interfejs.
    - Novodefinisane klase mogu slušati početak ili završetak izvršavanja testa, ili situacije u kojima testovi prolaze ili padaju
  - Report-era
    - Za implementaciju klase Reporter-a ptrebno je implementirati `org. testng.IReporter` interfejs.
    - Izveštaj biva kreiran nakon izvršavanja celog suite-a

logger/CustomLogging.java

src/simple-logger-testng.xml

CUSTOM LOGGER

- Custom Logger klasa implementiranjem `ITestListener` klase override-uje sve metode propisane interfejsom
- Vezivanje test klase sa novokreiranim loggerom se vrši u XML fajlu
- Moguće je test klasi dodeliti više od jednog listener-a
- Listener-i se uglavnom koriste kada želimo da dobijamo kontinuirane izveštaje prilikom izvršavanja testova

```
1 usage
10  public class CustomLogging implements ITestListener{
11
12      //Called when the test-method execution starts
        2 usages
13      @Override
14  @   public void onTestStart(ITestResult result) {
15          System.out.println("Test method started: "+ result.getName()+
16                  " and time is: "+getCurrentTime());
17      }
18      //Called when the test-method execution is a success
19      @Override
20  @   public void onTestSuccess(ITestResult result) {
21          System.out.println("Test method success: "+ result.getName()+
22                  " and time is: "+getCurrentTime());
23
24      }
25
26      //Called when the test-method execution fails
27      @Override
28  @   public void onTestFailure(ITestResult result) {
29          System.out.println("Test method failed: "+ result.getName()+
30                  "and time is: "+getCurrentTime());
31
32      }
```

simple-logger-testng.xml

```
1  <suite name="Simple Logger Suite">
2      <listeners>
3          <listener class-name="logger.CustomLogging" />
4      </listeners>
5      <test name="Simple Logger test">
6          <classes>
7              <class name="sample.SampleTest" />
8          </classes>
9      </test>
10 </suite>
```

CUSTOM REPORTER

- Custom Reportr klasa implementiranjem IReporter klase override-uje generateReport metodu
- Metod prima tri argumenta
  - Lista XML Suite-ova
  - Suite
  - outputDirectory
    - Sadrži informacije o putanji izlazne fascikle u kojoj će se generisati izveštaji

```
1 usage
public class CustomReporting implements IReporter {
    @Override
    public void generateReport(List<XmlSuite> xmlSuites,
                               List<ISuite> suites,
                               String outputDirectory) {
        //Iterating over each suite included in the test
        for (ISuite suite : suites) {
            //Following code gets the suite name
            String suiteName = suite.getName();
            //Getting the results for the said suite
            Map<String, ISuiteResult> suiteResults = suite.getResults();
            for (ISuiteResult sr : suiteResults.values()) {
                ITestContext tc = sr.getTestContext();

                System.out.println("Passed tests for suite '" + suiteName
                        + "' is:" + tc.getPassedTests().getAllResults().size());
                System.out.println("Failed tests for suite '" + suiteName
                        + "' is:" + tc.getFailedTests().getAllResults().size());
                System.out.println("Skipped tests for suite '" + suiteName
                        + "' is:" + tc.getSkippedTests().getAllResults().size());
            }
        }
    }
}
```

```
<suite name="Simple Reporter Suite">
    <listeners>
        <listener class-name="reporter.CustomReporting" />
    </listeners>
    <test name="Simple Reporter test">
        <classes>
            <class name="sample.SampleTest" />
        </classes>
    </test>
</suite>
```