Testarea claselor în JUNIT

Prof. univ. dr. ing. Florica Moldoveanu

Platforme de testare unitară

- JUnit: platforma (framework) de testare unitara pentru limbajul Java (http://www.junit.org).
- **Platforme similare**: Nunit (pentru C#), PyUnit (pentru Pyton), fUnit (pentru Fortran), CPPUnit (pentru C++), s.a., denumite in mod colectiv xUnit.

  http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

Platformele de testare unitară reduc efortul de testare permițând:

- Definirea cazurilor de test si organizarea lor în suite de teste
- Setarea contextului de execuție al cazurilor de test (operatii înainte şi după execuția cazurilor de test)
- Execuția suitelor de teste
- Inregistrarea rezultatelor executiei suitelor de teste
- Analiza rezultatelor testelor
- Reutilizarea suitelor de teste după modificarea codului – teste de regresie

Testarea claselor în JUNIT (1)

Pentru testarea unei clase C în Junit:

- Se implementează o **clasă de test**, TestC

  - Pentru fiecare metodă M a clasei C se implementează o metodă de test, TestM, în clasa TestC

    - Metoda TestM implementează unul sau mai multe cazuri de test ale metodei M:
      - apelează metoda M cu valori alese pentru parametrii de intrare
      - compară rezultatele întoarse de metoda M cu cele aşteptate, folosind aserțiuni

- Clasa **Assert** din mediul Junit furnizează metode de definire a aserțiunilor asupra rezultatelor aşteptate la execuția unei metode a clasei testate.

- Execuția unei metode aserțiune se termină fie normal, fie printr-o excepție, numită **AssertionFailedError**. **Exemplu de metodă aserțiune:**

```
static public void assertTrue(Boolean conditie) {
   if (!conditie)
            throw new AssertionFailedError();
}
```

Testarea claselor în JUNIT (2)

- O clasa de test in Junit 3 extinde clasa **TestCase** a mediului Junit.
- Functiile din clasa de test care sunt metode de test au nume prefixate cu “test”
- In Junit 4 metodele de test sunt adnotate cu @Test (https://www.guru99.com/junit-annotations-api.html)

```
public class Calcule {
   public static int Suma(int a, int b) {
      return (a+b);
   }
}
```

Junit 3

```
import junit.framework.*;// TestCase şi Assert
import Calcule;

public class TestCalcule extends TestCase {
   public void testSuma() {
     assertEquals(15, Calcule.Suma(5, 10));
  }
}
```

Junit 4

```
import org.junit.*;
import static org.junit.Assert.*;
import Calcule;

public class TestCalcule {
   @Test
   public void CalculSuma() {
     assertEquals(15, Calcule.Suma(5, 10));
   }
}
```

Definirea claselor de test în JUNIT 4 (1)

Exemplu:

Clasa de testat:

```
public class Find {
    public static int findMax(int arr[]){
     int max=0;
      for(int i=1; i < arr.length; i++){
        if(max<arr[i])  max=arr[i];
       }
       return max;
}
    public static int findMin(int arr[]){
       int min=0;
        for(int i=1; i <arr.length; i++){
          if(min>arr[i])  min=arr[i];
         }
        return min;
      }
  }
```

Clasa de test:

```
import org.junit.*;
import static org.junit.Assert.*;
import Find;
//2 cazuri de test pentru fiecare functie membru
public class TestFind {
   @Test
   public void testFindMax(){
    assertEquals (4, Find.findMax(new int[]{1,3,4,2}));
    assertEquals (-1,
        Find.findMax(new int[]{-12,-1,-3,-4,-2}));  }
   @Test
   public void testFindMin(){
    assertEquals (1, Find.findMin(new int[]{1,3,4,2}));
    assertEquals (-12,
        Find.findMin(new int[]{-12,-1,-3,-4,-2}));  }
}
```

Executia cazurilor de test în JUNIT 4 (1)

- Se creaza o clasa **TestRunner** si se apeleaza metoda runClasses() din clasa JUnitCore cu parametru numele clasei de test. Functia intoarce un obiect de tip *Result*, care poate fi utilizat pentru a obtine rezultatele testelor. Functia getFailures din clasa Result intoarce lista testelor care au cazut (public List<Failure> **getFailures**()).

```
import org.JUnit.runner.JUnitCore;
import org.JUnit.runner.Result;
import org.JUnit.runner.notification.Failure;
public class TestRunner {
  public static void main(String[] args) {
     Result result = JUnitCore.runClasses(TestFind.class);
     System.out.println("Caderi: " + result.getFailureCount());
     for (Failure failure: result.getFailures()) //pentru fiecare cadere
              { System.out.println(failure.toString()); }
 }
}
```

Executia cazurilor de test în JUNIT 4 (2)

- Se execută programul de test cu clasa **TestRunner:** se vor executa cazurile de test definite in clasa TestFind.
- Rezultatele afisate:

  Caderi: 2

  AssertionFailedError: expected: <-1> but was:<0>

  AssertionFailedError: expected: <1> but was:<0>

  Prima excepție este produsa de assertEquals (-1, Find.findMax(new int[]{-12,-1,-3,-4,-2}));

  A 2-a excepție este produsa de assertEquals (1, Find.findMin(new int[]{1,3,4,2}));

- Se înlocuiesc inițializările:

  max=0; cu max=arr[0];

  min=0; cu min=arr[0];
- Se ruleaza din nou **TestRunner**.
- Rezultatul afisat:

  Caderi: 0

Setarea contextului de executie al cazurilor de test în JUNIT 4 (1) - test fixture -

- Poate fi necesar ca anumite operatii sa fie executate înaintea unui caz de test/ după execuția cazului de test sau înaintea tuturor cazurilor de test definite în clasa de test/după execuția tuturor cazurilor de test.
- De exemplu, daca funcția testată lucrează cu un fişier care trebuie să existe atunci când este apelată. Crearea şi distrugerea fişierului nu fac parte din funcție.
  - Sunt 2 posibilități:
  - Crearea şi distrugerea fişierului să fie incluse în funcția de test (care implementează cazul de test al funcției) – nu se recomandă, deoarece aceste operații nu fac parte din funcția testată.
  - Crearea şi distrugerea fişierului să fie implementate separat de cazul de test, prin funcții speciale de setare a contextului de execuție a cazului de test.

Setarea contextului de executie al cazurilor de test în JUNIT 4 (2)

Alte exemple:

- Crearea unei conexiuni la o bază de date/ deconectarea după execuția cazului de test
- Pornirea unui server înaintea execuției tuturor cazurilor de test/ închiderea conexiunii după. Este ineficient ca pornirea şi oprirea serverului să fie implementate în fiecare caz de test.

- Astfel de operații pot fi definite în funcții membru ale clasei de test, adnotate cu:

  @Before: operații executate înaintea fiecărui caz de test

  @After: operații executate după fiecare caz de test

  @BeforeClass: operații executate înaintea tuturor cazurilor de test

  @AfterClass: operații executate după executia tuturor cazurilor de test

Setarea contextului de executie al cazurilor de test în JUNIT 4 (3)

Exemplu:

```
public class OutputFileTest { //clasa de test
     private File output;
    @BeforeClass
      public static void alocari(){ …aloca resurse pentru toate testele…}
    @AfterClass
      public static void dealocari(){ …dealoca resursele folosite pentru toate testele…}
    @Before
      public void createOutputFile(){ output = new File (…); }
    @After
      public void deleteOutputFile() { output.delete(); }
    @Test
       public void testFile1() { ….cazul de test 1 pt lucrul cu fisierul output…}
    @Test
       public void testFile2() { ….cazul de test 2 pt lucrul cu fisierul output…}
}
```

Setarea contextului de executie al cazurilor de test în JUNIT 4 (4)

Ordinea la executie a metodelor clasei OutputFileTest:

```
alocari();
createOutputFile();
testFile1();
deleteOutputFile();
createOutputFile();
testFile2();
deleteOutputFile();
dealocari();
```

Rularea unei suite de teste în JUNIT 4

Testele (cazurile de test) definite în mai multe clase de test pot forma o suita de teste.

- Se creaza o clasa adnotata cu **@RunWith** şi **@Suite** si se adauga clasele de test la suita.
- Clasa astfel adnotata nu are cod – este doar un suport pentru adnotari.

```
import org.JUnit.runner.*;

@Runwith(Suite.class)

@Suite.SuiteClasses({test1.class, test2.class……})
 public class TestSuiteExample { } //clasa nu are cod
 public class TestRunner { //rularea testelor definite in clasele din suita (test1, test2,..)
    public static void main(String[] args) {
     Result result = JUnitCore.runClasses(TestSuiteExample.class);
     System.out.println("Caderi: " + result.getFailureCount());
     for (Failure failure: result.getFailures()) { System.out.println(failure.toString()); }
 }
}
```

Functii din clasa Assert

- ***assertEquals(Object asteptat, Object actual):*** executia metodei se termina normal daca obiectul asteptat si cel actual sunt egale conform metodei **equals()**: asteptat**.equals(** actual **) ==** true
- ***assertEquals(int asteptat, int actual):*** se termina normal daca *asteptat* **==** *actual*

  Exista o asertiune similara pentru fiecare tip primitiv: int, float, double, char, byte, long, short, Boolean şi String.
- ***assertEquals(double asteptat, double actual, double toleranţa*)**: metoda se termina normal daca valoarea absoluta a diferentei dintre valoarea asteptata si cea actuala este <= cu valoarea de toleranta; exista o asertiune similara pentru intrari float.
- ***assertSame(Object asteptat, Object actual)*:** metoda se termina normal daca parametrii *asteptat* si *actual* sunt referinte la acelasi obiect din memorie.
- ***assertNull(Object testobject):*** se termina normal daca *testobject* este *null*
- ***assertFalse(Boolean conditie):*** se termina normal daca expresia *conditie* are valoarea FALSE.
- ***assertTrue(Boolean conditie):*** se termina normal daca expresia *conditie* are valoarea TRUE.

Lecturi suplimentare

1. https://www.guru99.com/junit-annotations-api.html
2. https://junit.org/junit4/javadoc/latest/org/junit/runner/JUnitCore.html#runClasses(java.lang.Class...)
3. https://junit.org/junit4/javadoc/latest/org/junit/runner/Result.html
4. https://www.javatips.net/api/org.junit.runner.result