Testiranje softvera

Vežbe - JUnit verzija 5
Alat za jedinično testiranje

Dražen Drašković, docent
Elektrotehnički fakultet
Univerziteta u Beogradu

Testiranje Java programa uz pomoć alata JUnit

- JUnit je framework predviđen za jedinično testiranje pojedinačnih java klasa ili grupe klasa.
- Filozofija koja se promoviše uz JUnit je test-first projektovanje, da sam programer paralelno sa kodiranjem piše i testove.
- JUnit može da se koristi i u klasičnom testiranju za automatizaciju testiranja.

JUnit biblioteka

- JUnit biblioteka: junit-5.jar
  (link za download: https://junit.org/junit5/)
- Potrebno je da JAR fajl dodate u putanju projekta

Prednosti JUnit-a

- Odvaja instance test klase za svaki jedinični test
- Specifična JUnit anotacija koja obezbeđuje inicijalizaciju i redefinisanje metoda: @Before, @BeforeClass, @After, @AfterClass
- Veliki broj različitih metoda za proveru rezultata našeg testa
- Integracija sa popularnim Java aplikacijama: Eclipse, NetBeans, IntelliJ, JBuilder

Pisanje JUnit test skriptova

- Kao primer, želimo da testiramo klasu Calculator, koja operiše celim brojevima i čuva rezultat u statičkoj promenljivoj result.
- Da bi primer bio realističan, subtract operacija ima grešku, množenje nije još implementirano, a kod korenovanja program upada u “mrtvu petlju”.
- Kôd primera se nalazi na deljenom disku.

```
package calc;
public class Calculator {
    private static int result;    // Statička promenljiva za smeštaj rezultata
    public void add(int n) {
        result = result + n;
    }
    public void substract(int n) {
        result = result - 1;      // Greška: trebalo bi result = result - n
    }
    public void multiply(int n) {
    }          // nije jos spremno za testiranje
    public void divide(int n) {
        result = result / n;
    }
    public void square(int n) {
        result = n * n;
    }
    public void squareRoot(int n) {
        for (; ;) ;                 // Greška: mrtva petlja
    }
    public void clear() {            // Postavlja rezultat na nulu
        result = 0;
    }
    public void switchOn() {     // Početak korišćenja kalkulatora
        result = 0;
    }
    public void switchOff() {     // Kraj korišćenja kalkulatora
    }
    public int getResult() {
        return result;
    }
}
```

Pisanje JUnit test skripta

- Kod za testiranje se piše u javi
- Za klasu test napisaćemo klasu CalculatorTest, pojedini metodi biće konkretni testovi
- Pretpostavimo da su klase Calculator i CalculatorTest smeštene u isti paket

CalculatorTest.java

```
package calc;
import static org.junit.Assert.*;
import org.junit.*;
/** Basic test class using @Test, @Before and @Ignore annotation
 *  as well as assert keyword and assertEqual methods
 */
public class CalculatorTest {

    private static Calculator calculator = new Calculator();

    @Before
    // must be public not protected like the setup
    public void clearCalculator() {
        calculator.clear();
    }
    //==================================
    //=            Test cases          =
    //==================================
    @Test
    public void add() {
        calculator.add(1);
        calculator.add(1);
        assertEquals(2, calculator.getResult());
    }
```

```
 @Test
    public void subtract() {
        calculator.add(10);
        calculator.substract(2);
        assertEquals(8,calculator.getResult());
    }

    @Test
    public void divide() {
        calculator.add(8);
        calculator.divide(2);
        assertEquals(5,calculator.getResult());
    }

    @Test(expected = ArithmeticException.class)
    public void divideByZero() {
        calculator.divide(0);
    }

    // @Ignore has a String parameter which displays a message
    @Test
    @Ignore("not ready yet")
    public void multiply() {
        calculator.add(10);
        calculator.multiply(10);
        assertEquals(100,calculator.getResult());
    }
}
```

Pisanje JUnit test skripta

- Na početku skripta treba importovati:

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

- import static služi da se provere (asserts) pišu bez prefiksa klase (to su u stvari statički metodi klase Assert iz JUnita)
- drugi import služi da bi bile vidljive klase JUnit frameworka kao što su Test itd.
- Klasa koju pišemo za testiranje ne treba ništa da nasleđuje, kao što je bio slučaj u JUnit 3.x (ovde se koristi java mehanizam refleksije)

Pisanje jednog test primera: @Test

- Svaki poseban test piše se kao bezparametarski javni void metod u test klasi, koji mora da ima prefiks **@Test**:

```
@Test
    public void add() {
      calculator.add(1);
      calculator.add(1);
      assertEquals(2,calculator.getResult());
    }
```

Provera rezultata testa: assert

- Rezultat izvršavanja koda koji se testira proverava se unutar test metoda pozivom jednog ili više assert metoda definisanih u okviru alata JUnit:
- ```
  @Test
   public void add() {
     calculator.add(1);
     calculator.add(1);
     assertEquals(2,calculator.getResult());
   }
  ```
  ```

Provera rezultata testa: assert

- assertEquals(expected, actual)

Očekivana vrednost

Vrednost koja se računa u kodu koji se testira

- Ako ne dođe do poklapanja, assert baca izuzetak **java.lang.AssertionError** i prekida izvršavanje tekućeg test metoda, ali ne i ostalih.

Provera rezultata testa: assert

- assertEquals(message, expected, actual)
- assertEquals(expected, actual, delta),
- assertEquals(message, expected, actual, delta)
  - Upotrebljava se za double ili float, gde delta predstavlja prihvatljivo odstupanje izmedju očekivane i sračunate vrednosti
- assertFalse(condition),
- assertFalse(message,condition)
  - Provera da li je uslov false
- assertTrue(condition), assertTrue(message,condition)
  - Provera da li je uslov true
- fail(), fail(message)
  - Prekida test i javlja da test nije prošao

Pripremne i završne radnje (fixture)

- Često imamo dva ili više testova koji koriste isti skup objekata
  - Ovaj skup objekata naziva se test fixture
- Upotreba test fixture-a:
  - Kreirati člana klase za svaki objekat iz test fixture
  - Napisati metod koji postavlja skup radnih objekata (staviti oznaku @Before)
  - Napisati metod koji čisti skup radnih objekata (staviti oznaku @After)

Pripremne i završne radnje (fixture)

- Primer:

```
public class CalculatorTest {

    private static Calculator calculator=new
    Calculator();

    @Before
    public void clearCalculator() {
        calculator.clear();
    }
. . .
```

Pripremne i završne radnje (fixture)

- Metode obeležene sa @Before se pozivaju pre svakog test metoda
- Metode obeležene sa @After se pozivaju nakon svakog metoda
- Zašto?
- Izbegavanje bočnih efekata između pokretanja test metoda

Testiranje izuzetaka

- Oznaka @Test dopušta upotrebu opcionog **expected** parametra.
- Njime se deklariše da se očekuje da test baci određeni izuzetak. Ako to ne uradi, ili baci drugačiji izuzetak, test tada nije prošao.

```
@Test(expected = ArithmeticException.class)
    public void divideByZero() {
        calculator.divide(0);
    }
```

Deaktiviranje testa

- Ako želimo da privremeno preskočimo izvršavanje nekog testa (a da se u finalnom izveštaju to pojavi kao ignorisani test), koristi se oznaka @Ignore:

```
@Test
@Ignore("not ready yet")
  public void multiply() {
    calculator.add(10);
    calculator.multiply(10);
    assertEquals(100,calculator.getResult());
  }
```

- U opcionom string parametru @Ignore, navodi se razlog zašto je test primer deaktiviran.

Izvršavanje testova

- Varijanta iz komandne linije: prevesti kod koji se testira, samu test klasu i pokrenuti test runner klasu JUnita
- Dodati JUnit biblioteku u projekat i izvršiti u okviru programerskog okruženja (Eclipse)

Rezultati izvršavanja

```
JUnit version 5
...E.EI
Time: 0.078
There were 2 failures:
1) subtract(calc.CalculatorTest)
java.lang.AssertionError: expected:<9> but was:<8>
      at org.junit.Assert.fail(Assert.java:91)
2) divide(calc.CalculatorTest)
java.lang.AssertionError: expected:<4> but was:<5>
      at org.junit.Assert.fail(Assert.java:91)

FAILURES!!!
Tests run: 4,  Failures: 2
```

- U drugoj liniji svaki test koji je uspešno prošao ispisuje tačku, onaj koji je neuspešan E, a ignorisani I.

Pisanje parametrizovanih testova

- Ako želimo da izvršimo određeni test metod više puta, sa različitim ulaznim veličinama:

1. Navodi se klasa za izvršavanje testova Parameterized.
2. Sama klasa za testiranje (koja sadrži test metode) mora imati public static metod koji je označen sa @Parameters i vraća Collection objekat sa skupovima parametara
3. Klasa za testiranje mora imati i javni konstruktor koji prihvata ove parametre.

- Tokom izvršavanja testova, klasa za testiranje biće instancirana onoliko puta koliko u Collection objektu ima skupova parametara

Primer parametrizovanog testa

```
package calc;

import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

import java.util.*;

@RunWith(Parameterized.class)
/**
 * This class is a parameterized test case. It uses the
 * values defined in the collection as a parameter.
 */
public class SquareTest {

   private static Calculator calculator = new Calculator();
   private int param;
   private int result;
```

Squaretest.java

```
@Parameters
   public static List param() {
      return Arrays.asList(new Object[][]{
            {0, 0},
            {1, 1},
            {2, 4},
            {4, 16},
            {5, 25},
            {6, 36},
            {7, 48}        // 7² = 49 not 48
      });
   }

   public SquareTest(int param, int result) {
      this.param = param;
      this.result = result;
   }

   //===================================
   //=          Test cases             =
   //===================================
   @Test
   public void square() {
      calculator.square(param);
      assertEquals(result, calculator.getResult());
   }
```

Izvršavanje parametrizovanog testa

- Rezultat izvršavanja:

```
JUnit version 5
.......E
Time: 0.063
There was 1 failure:
1) square[6](calc.SquareTest)
java.lang.AssertionError: expected:<48> but was:<49>
    at org.junit.Assert.fail(Assert.java:91)

FAILURES!!!
Tests run: 7,  Failures: 1
```

Kolekcija testova (test suite)

- Ako želimo istovremeno da pokrenemo testove iz klasa CalculatorTest i SquareTest, kreiraćemo kolekciju testova **AllCalculatorTests.java**:

```
package calc;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
      CalculatorTest.class,
      SquareTest.class
      })
/**
 * This class is a suite of two other test cases
 */
public class AllCalculatorTests {
}
```

Određuje JUnit klasu odgovornu za izvršavanje testova

Određuje koje test klase treba uključiti u kolekciju testova AllCalculatorTests

Pokretanje kolekcije testova

- Izlaz:

```
JUnit version 5
...E.EI.......E
Time: 0.109
There were 3 failures:
1) subtract(calc.CalculatorTest)
java.lang.AssertionError: expected:<9> but was:<8>
      at org.junit.Assert.fail(Assert.java:91)
2) divide(calc.CalculatorTest)
java.lang.AssertionError: expected:<4> but was:<5>
      at org.junit.Assert.fail(Assert.java:91)
3) square[6](calc.SquareTest)
java.lang.AssertionError: expected:<48> but was:<49>
      at org.junit.Assert.fail(Assert.java:91)

FAILURES!!!
Tests run: 11,  Failures: 3
```

Preporučena literatura

- **JUnit 5, zvanična dokumentacija**