

## Testarea unitară a claselor Java folosind utilitarul JUnit

Testarea unitară s-a impus în ultima perioadă în dezvoltarea proiectelor scrise în limbajul Java și numai, pe măsura apariției unor utilitare gratuite de testare a claselor, care au contribuit la creșterea vitezei de programare și la micșorarea drastică a numărului de bug-uri.

Cel mai folosit utilitar pentru testarea unitară a claselor Java este JUnit, care se poate descărca gratuit de pe site-ul <http://www.junit.org>. Arhiva este destul de mică (aproximativ 400 Kb) și include un director (junitxxx) cu documentație (în directorul doc), documentația API (în directorul javadoc), biblioteca de clase junit.jar și exemple de clase de test (în directorul junit).

Printre avantajele folosirii utilitarului JUnit se numără:

- se îmbunătățește viteza de scriere a codului, concomitent cu creșterea calității acestuia, deoarece prin scrierea testelor unitare se micșorează timpul de depanare, pemițând refactorizarea mai ușoară, cu depistarea imediată a eventualelor erori inserate în codul modificat;
- 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. Compilatorul testează sintaxa codului sursă, în timp ce clasele de test validează integritatea codului
- clasele de test JUnit pot fi rulate automat (în suită), rezultatele fiind vizibile imediat. Se pot crea ierarhii de suite de test, care pot fi testate împreună sau separat, în funcție de cerințele proiectului
- clasele de test măresc încrederea programatorului în codul sursă scris și îi permit să urmărească mai ușor cerințele de implementare ale proiectului, putând constitui și o parte a documentației finale transmise clientului
- JUnit este un utilitar gratuit, iar testele JUnit sunt scrise în Java și beneficiază de portabilitatea acestuia

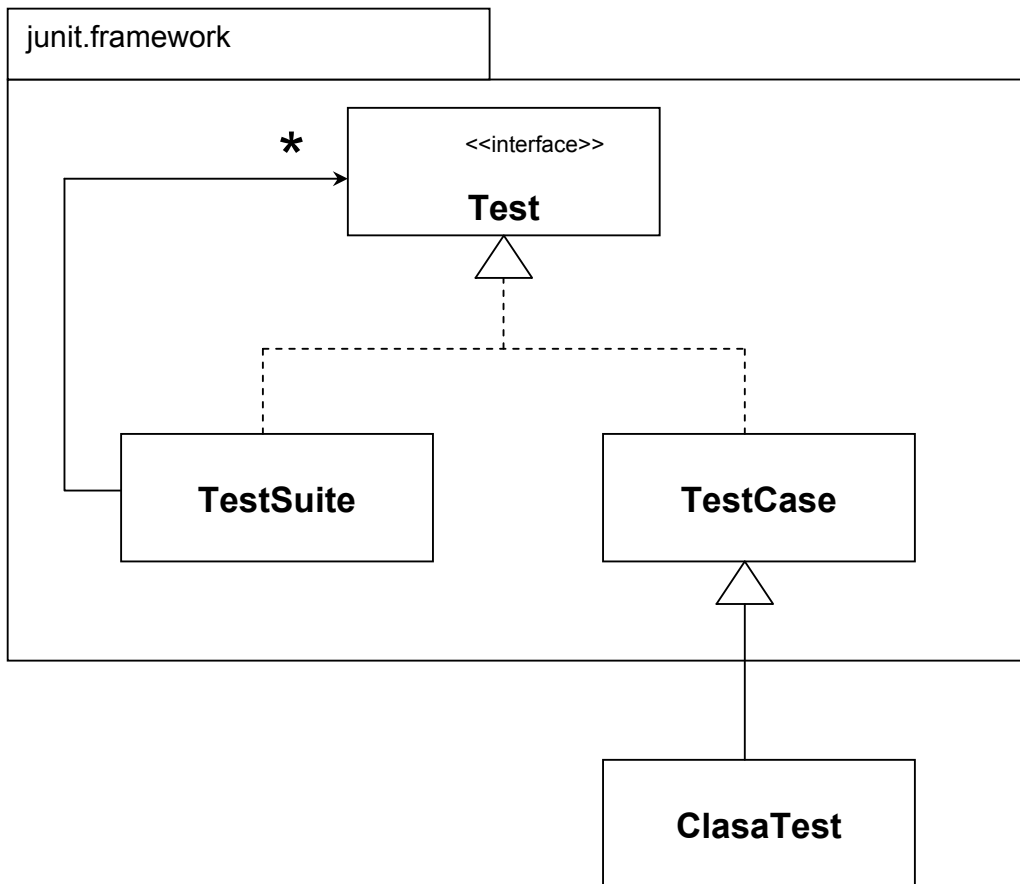
JUnit a fost proiectat pe baza a două modele (patterns): modelul Command și modelul Composite.

O clasă TestCase este un obiect command și orice clasă ce conține metode de test trebuie să subclaseze clasa TestCase. O clasă TestCase se compune dintr-un număr de metode publice testXXX(). Pentru a verifica rezultatele așteptate și cele curente se va invoca una dintre metodele assert().

Metodele setUp() și tearDown() servesc la initializarea și distrugerea oricărui obiect utilizat în cadrul testelor. Fiecare test rulează într-un context propriu și apelează metoda setUp() înainte și metoda tearDown() după fiecare metodă de test pentru a evita efectele secundare dintre teste.

Instanțele TestCase se pot compune sub forma unor ierarhii TestSuite ce vor invoca automat toate metodele testXXX() definite în fiecare instanță TestCase. O clasă TestSuite poate fi compusă din alte teste, instanțe TestCase sau alte instanțe TestSuite.

Diagrama de clase a pachetului junit.framework este următoarea:



## Exemple de clase de test

Să considerăm ca exemplu o aplicație de tip librărie virtuală, în care se definește o clasă `Book` ce păstrează informații despre cărțile din librărie și o clasă de tip coș de cumpărături `ShoppingCart` ce păstrează lista cărților pe care un client dorește să le cumpere.

Codul sursă al celor două clase este prezentat mai jos:

## Clasa Book.java

```
/**
 * Un model de carte pentru testare cu JUnit.
 *
 */
public class Book {

    private String _title;
    private String _author;
    private double _price;

    /**
     * Constructor pentru <code>Book</code>.
     *
     * @param title Book title.
     * @param author Book author.
     * @param price Book price.
     */
    public Book(String title, String author, double price) {
        _title = title;
        _author = author;
        _price = price;
    }

    /**
     * Returneaza titlul cartii.
     *
     * @return Title.
     */
    public String getTitle() {
        return _title;
    }

    /**
     * Returneaza autorul cartii.
     *
     * @return Author.
     */
    public String getAuthor() {
        return _author;
    }

    /**
     * Returneaza pretul cartii.
     *
     * @return Price.
     */
    public double getPrice() {
        return _price;
    }

    /**
     * Testare egalitate intre obiecte Book.
     *
     * @return <code>true</code> daca obiectele sunt egale.
     */
    public boolean equals(Object obj) {

        if (obj instanceof Book) {
            Book b = (Book)obj;
            return b.getTitle().equals(_title);
        }

        return false;
    }
}
```

## Clasa ShoppingCart.java

```
import java.util.*;

/**
 * Un exemplu de clasa de tip cos de cumparaturi.
 *
 */

public class ShoppingCart {

    private ArrayList _items;

    /**
     * Constructor pentru o instanta <code>ShoppingCart</code>.
     */
    public ShoppingCart() {
        _items = new ArrayList();
    }

    /**
     * Returneaza pretul total.
     *
     * @return double.
     */
    public double getTotal() {
        Iterator i = _items.iterator();
        double total = 0.00;
        while (i.hasNext()) {
            Book book = (Book)i.next();
            total = total + book.getPrice();
        }

        return total;
    }

    /**
     * Aadauga o carte in cos.
     *
     * @param book Book.
     */
    public void addItem(Book book) {
        _items.add(book);
    }

    /**
     * Scoate o carte din cos.
     *
     * @param book Book.
     * @throws Exception daca acea carte nu exista.
     */
    public void removeItem(Book book) throws Exception {
        if (!_items.remove(book)) {
            throw new Exception();
        }
    }

    /**
     * Returneaza numarul de carti din cos.
     *
     * @return Item count.
     */
    public int getItemCount() {
        return _items.size();
    }

    /**
     * Goleste cosul.
     */
    public void empty() {
        _items = new ArrayList();
    }
}
```

```
/**
 * Indica faptul ca s-a golit cosul.
 *
 * @return <code>true</code> daca e un cos gol;
 *         <code>false</code> in caz contrar.
 */
public boolean isEmpty() {
    return (_items.size() == 0);
}
}
```

Etapele creării unei clase de test sunt:

- se definește o subclasă a clasei `TestCase`;
- se suprascrie metoda `setUp()` pentru inițializarea obiectelor testate
- se suprascrie metoda `tearDown()` pentru eliberarea obiectelor testate
- se definește una sau mai multe metode publice `testXXX()` ce testează obiectele și anticipează rezultatele așteptate
- se definește o metodă statică `suite()` de tip factory ce creează o suită `TestSuite` care conține toate metodele `testXXX()` din subclasa clasei `TestCase`
- opțional se definește o metodă `main()` ce returnează subclasa `TestCase` în mod batch

Un exemplu de clasă de test pentru coșul de cumpărături este prezentat mai jos:

## Clasa ShoppingCartTest.java

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Clasa <code>ShoppingCartTest</code> este un exemplu de
 * clasa <code>TestCase</code>.
 *
 */

public class ShoppingCartTest extends TestCase {

    private ShoppingCart _bookCart;
    private Book _aBook;

    /**
     * Constructor pentru o instanta <code>ShoppingCartTest</code> cu numele
     * specificat.
     *
     * @param name Nume test.
     */
    public ShoppingCartTest(String name) {
        super(name);
    }

    /**
     * Initializare obiecte de test.
     *
     * Se apeleaza inaintea fiecarei metode de test.
     */
    protected void setUp() {

        _bookCart = new ShoppingCart();
        _aBook = new Book("Unit testing","John Smith", 35.95);
        _bookCart.addItem(_aBook);
    }

    /**
     * Distruge obiectele testate.
     *
     * Se apeleaza dupa fiecare metoda de test.
     */
    protected void tearDown() {
        _bookCart = null;
    }

    /**
     * Testare inserare carti in cos.
     */
    public void testBookAdd() {

        Book book = new Book("Extreme machines","Thomas Manner", 46.95);
        _bookCart.addItem(book);
        double expectedTotalPrice = _aBook.getPrice() + book.getPrice();

        assertEquals(expectedTotalPrice, _bookCart.getTotal(), 0.0);
        assertEquals(2, _bookCart.getItemCount());
    }

    /**
     * Test golire cos.
     */
    public void testEmpty() {

        _bookCart.empty();
        assertTrue(_bookCart.isEmpty());
    }
}
```

```

/**
 * Testare scoatere carti din cos.
 *
 * @throws Exception Daca nu exista acea carte in cos.
 */
public void testBookRemove() throws Exception {

    _bookCart.removeItem(_aBook);
    assertEquals(0, _bookCart.getItemCount());
    assertEquals(0.0, _bookCart.getTotal(), 0.0);
}

/**
 * Testare scoatere carte inexistentă din cos.
 */
public void testBookNotFound() {

    try {

        Book book = new Book("Toy story", "John Scott" ,54.95);
        _bookCart.removeItem(book);

        fail("Trebuie sa semnaleze o exceptie");

    } catch(Exception w) {
    }
}

/**
 * Asambleaza si returneaza o suita de teste pentru
 * toate metodele de test din calasa de test
 *
 * @return O clasa <code>Test</code> nenula.
 */
public static Test suite() {

    //
    // Adaugarea tuturor metodelor de forma testXXX()
    // la suita folosind Java Reflection
    //
    TestSuite suite = new TestSuite(ShoppingCartTest.class);

    //
    // Alternativ, se poate adauga cate o metoda de test...
    //
    // TestSuite suite = new TestSuite();
    // suite.addTest(new ShoppingCartTest("testBookAdd"));
    // suite.addTest(new ShoppingCartTest("testEmpty"));
    // suite.addTest(new ShoppingCartTest("testBookRemove"));
    // suite.addTest(new ShoppingCartTest("testBookNotFound"));
    //

    return suite;
}

/**
 * Main.
 */
public static void main(String args[]) {
    junit.textui.TestRunner.run(suite());
    //junit.swingui.TestRunner.run(ShoppingCartTest.class);
}
}

```

Se observă că clasa `ShoppingCartTest` este extinsă din clasa `TestCase`. S-a inițializat în metoda `setUp()` un obiect `_bookCart` de clasă `ShoppingCart` și un obiect `_aBook` de clasă `Book`, care a fost adăugat obiectului `_bookCart`.

În metoda `testBookAdd()` s-a testat adăugarea unei cărți în coș folosind metoda `assertEquals(double expected, double actual, double delta)`, delta fiind diferența acceptată pentru comparații între două valori de tip `double`, precum și incrementarea numărului de itemuri din coș cu metoda `assertEquals(int expected, int actual)`

Metoda `testEmpty()` testează golirea coșului după apelul metodei `empty()`. S-a utilizat metoda `assertTrue(boolean condition)` care afișează un mesaj de eroare în cazul în care condiția este falsă.

Metoda `testBookRemove()` testează operația de eliminare a unui obiect `Book` din coșul `_bookCart`. Coșul are inițial un obiect `Book` (inserat prin metoda `setUp()`) care este eliminat din coș cu metoda `removeItem()`. Se testează ca numărul de obiecte `Book` rămase să fie zero.

Metoda `testBookNotFound()` testează inexistența în coș a unui obiect `Book` care nu a fost adăugat anterior, în caz de eroare se aruncă o excepție prin apelul metodei `fail()`.

Pentru testare se adaugă la `CLASSPATH` calea către biblioteca `junit.jar` și către directorul unde se află clasele sursă, se compilează toate cele trei clase și se lansează în execuție cu comanda:

```
java ShoppingCartTest (din directorul claselor)
```

Rezultatul la consolă este prezentat în imaginea următoare:



```
D:\projects\JUnit\bin>java ShoppingCartTest
---
Time: 0,06
OK (4 tests)
D:\projects\JUnit\bin>
```

JUnit furnizează atât o interfață text pentru utilizator, cât și o interfață grafică, fiecare indicând numărul de teste, erorile și starea finală a testului.

Interfața text (`junit.textui.TestRunner`) afișează câte un punct pentru fiecare test efectuat și un OK dacă toate testele au fost trecute cu succes sau mesaje de eroare dacă unul dintre teste a eșuat.

Interfața grafică (`junit.swingui.TestRunner`) afișează o fereastră de tip Swing cu un bară verde dacă toate testele au trecut cu succes sau o bară roșie dacă unul sau mai multe teste eșuează.

Interfața grafică este specificată în metoda `main()`.

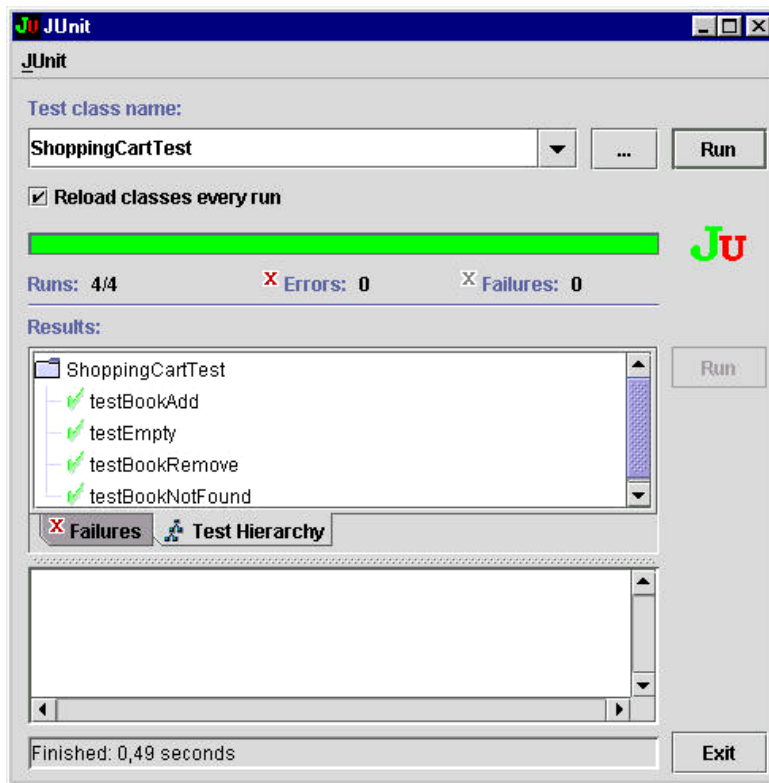
Pentru a afișa rezultatele testelor folosind interfața grafică, în metoda `main()` în loc de

```
junit.textui.TestRunner.run(suite());
```

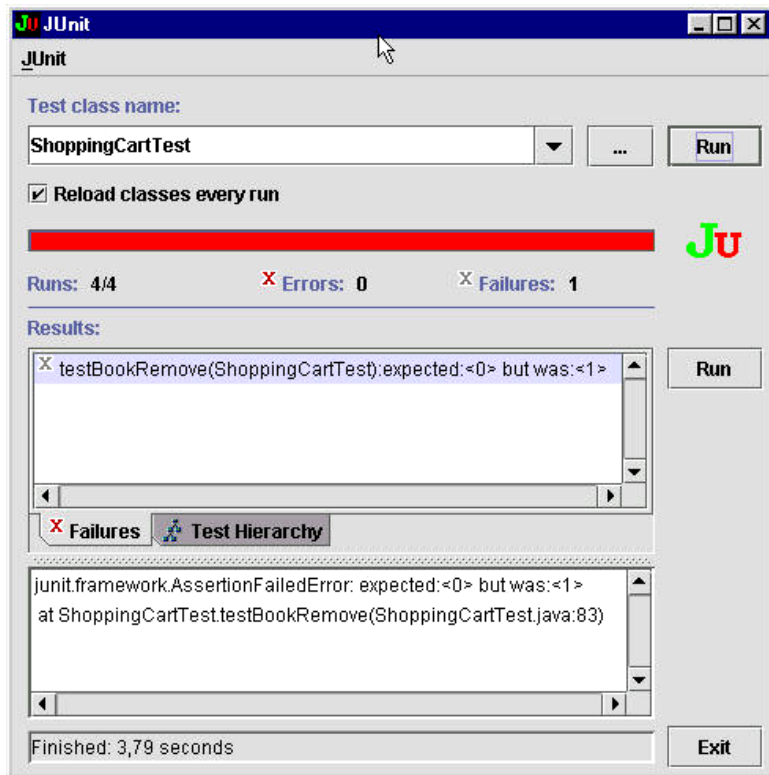
se scrie

```
junit.swingui.TestRunner.run(ShoppingCartTest.class);
```

Se recompilă clasa de test și se lansează în execuție, rezultatul afișat fiind prezentat în imaginea următoare:



Dacă un anumit test eșuează (de exemplu operația de eliminare a unui obiect Book din ShoppingCart), atunci se va afișa un mesaj de eroare de forma:



Metodele assertXXX() utilizate la scrierea testelor sunt prezentate în tabelul de mai jos:

<code>assertEquals(primitive expected, primitive actual)</code>	Verifică dacă cele două obiecte primitive sunt egale
<code>assertEquals(Object expected, Object actual)</code>	Verifică dacă cele două obiecte sunt egale (folosind metoda <code>equals()</code> din <code>Object</code> )
<code>assertSame(Object expected, Object actual)</code>	Verifică dacă cele două obiecte au aceeași adresă de memorie
<code>assertNotSame(Object expected, Object actual)</code>	Verifică dacă cele două obiecte nu sunt unul și același (nu au aceeași adresă de memorie)
<code>assertNull(Object object)</code>	Verifică dacă un obiect este nul
<code>assertNotNull(Object object)</code>	Verifică dacă un obiect nu este nul
<code>assertTrue(boolean condition)</code>	Verifică dacă o condiție este adevărată
<code>assertFalse(boolean condition)</code>	Verifică dacă o condiție nu este adevărată

Se pot crea suite de test ce include o ierarhie de clase de test sau alte suite. Etapele creării unei suite de test sunt:

- se definește o subclasă a clasei `TestCase`
- se definește o metodă factory statică `suite()` ce creează o instanță `TestSuite` ce conține toate testele
- opțional se definește o metodă ce lansează instanța `TestSuite` în mod batch

Un exemplu de clasă suită de test este următorul:

## Clasa AllTests

```
import junit.framework.Test;
import junit.framework.TestSuite;

public class AllTests {

    /**
     * Asambleaza si returneaza o suita de teste
     * Se pot adauga teste noi.
     *
     * @return O suita de teste.
     */
    public static Test suite() {

        TestSuite suite = new TestSuite();

        //
        // clasa ShoppingCartTest.
        //
        suite.addTest(ShoppingCartTest.suite());

        //
        // O alta clasa de test.
        //
        suite.addTest(CreditCardTest.suite());

        return suite;
    }

    /**
     * Lanseaza in executie suita de teste.
     */
    public static void main(String args[]) {
        junit.swingui.TestRunner.run(AllTests.class);
    }
}
```

Pentru testarea suitei se compilează toate clasele și se lansează comanda  
java AllTests

## Testarea automată folosind utilitarul Ant

Dacă proiectul de implementat este construit folosind utilitarul Ant (<http://ant.apache.org/>), atunci în fișierul build.xml se pot insera teste unitare. Să presupunem că sursele se află în directorul src, clasele generate în directorul bin, iar biblioteca junit.jar se află în directorul lib al proiectului. Se pot compila sursele Java și se pot rula testele folosind fișierul build.xml, plasat în directorul rădăcină al proiectului. Biblioteca junit.jar trebuie să se găsească în directorul lib al directorului unde este instalat Ant.

Conținutul fișierului build.xml este prezentat mai jos:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<project name="sample-junit" default="all" basedir=".">
<description>Build Sample JUnit project</description>

<property name="src" value="src"/>
<property name="lib" value="lib"/>
<property name="tmp" value="tmp"/>
<path id="cp">
  <fileset dir="${lib}" includes="*.jar"/>
  <pathelement path="${tmp}"/>
</path>
```

```

<target name="bin" description="Compile Java source files">
  <javac srcdir="${src}" destdir="${tmp}" debug="on" deprecation="on"
    classpathref="cp"/>
</target>

<target name="test" depends="bin" description="Run JUnit tests">
  <junit haltonfailure="false" haltonerror="false" printsummary="withOutAndErr">
    <classpath refid="cp"/>
    <batchtest>
      <fileset dir="${src}" includes="**/*Test*.java"/>
    </batchtest>
  </junit>
</target>

<target name="clean" description="Clean generated files">
  <delete dir="${tmp}"/>
  <mkdir dir="${tmp}"/>
</target>

<target name="all" depends="clean,test" description="Build the whole project"/>
</project>

```

S-au folosit opțiunile `haltonfailure=false` și `haltonerror=false` pentru a nu opri execuția testului în cazul apariției unei erori.

După lansarea testului cu comanda `ant` în rădăcina directorului proiectului, se obțin mesaje:

```

D:\projects\JUnit>ant
Buildfile: build.xml

clean:
  [delete] Deleting directory D:\projects\JUnit\tmp
  [mkdir] Created dir: D:\projects\JUnit\tmp

bin:
  [javac] Compiling 3 source files to D:\projects\JUnit\tmp

test:
  [junit] Running ShoppingCartTest
  [junit] Tests run: 4, Failures: 0, Errors: 0, Time elapsed: 0 sec

all:

BUILD SUCCESSFUL
Total time: 33 seconds
D:\projects\JUnit>

```

Ant poate afișa rapoarte de test în format HTML. Pentru aceasta trebuie înlocuit targetul de test cu cel de mai jos:

```

<target name="test" depends="bin"
  description="Run JUnit tests">
  <junit haltonfailure="false"
    printsummary="withOutAndErr">
    <classpath refid="cp"/>
    <batchtest todir="${tmp}">
      <fileset dir="${src}"
        includes="**/*Test*.java"/>
    </batchtest>
    <formatter type="xml"/>
  </junit>
  <junitreport todir="${tmp}">
    <fileset dir="${tmp}"
      includes="TEST-*.xml"/>
  <report format="frames"

```

```
        todir="${tmp}"/>
    </junitreport>
</target>
```

Ant va genera un raport XML în directorul tmp, iar elementul `junitreport` va genera un raport HTML din fișierele XML. Elementul `junitreport` necesită instalarea bibliotecii Xalan versiunea 2 în directorul lib al instanței Ant. Raportul generat arată ca în imaginea următoare:

The screenshot shows the 'Unit Test Results' page for the class 'ShoppingCartTest'. It includes a summary table and a detailed tests table.

Name	Tests	Errors	Failures	Time(s)
ShoppingCartTest	4	0	0	1.700

Name	Status	Type	Time(s)
testBookAdd	Success		0.050
testEmpty	Success		0.000
testBookRemove	Success		0.000
testBookNotFound	Success		0.000

Acest gen de raport este util dacă se rulează un număr mare de teste simultan.

Câteva recomandări cu privire la organizarea și execuția testelor:

- se vor crea clase de test în același pachet ca și codul testat
- pentru a evita combinarea codului de test și de aplicație în directoarele sursă, se va crea o structură de directoare identică (mirrored) cu structura pachetului ce conține codul de test
- pentru fiecare pachet Java al aplicației se va defini o clasă TestSuite ce conține toate testele pentru validarea codului din pachet
- se va verifica faptul că procesul de construire (build) al proiectului include compilarea tuturor testelor, pentru a se asigura că testele apelează ultima variantă a codului sursă
- se va alterna scrierea codului sursă cu operații de testare pentru a se depista rapid eventuale bug-uri
- se vor scrie teste pentru zonele cu risc maxim de erori
- dacă un bug este raportat în aplicația productivă, se va scrie un test care expune acest bug
- se vor scrie teste unitare înaintea scrierii codului și se va scrie cod nou numai când un test eșuează.

## Concluzie

JUnit este un utilitar valoros oricărui programator Java, permițând creșterea vitezei de programare și depistarea bug-urilor încă în faza de implementare a codului sursă. Mai multe unelte de dezvoltare pentru Java (cum ar fi JBuilder sau Eclipse) au încorporat suport pentru JUnit și Ant.

Pentru mai multe informații vă invit să vizitați site-ul web <http://www.junit.org> ce conține multe articole și documentație despre JUnit.

Cei care lucrează și cu alte limbaje de programare pot folosi alte utilitare de testare unitară ce se pot descărca gratuit de pe Internet:

CPPUnit - <http://cppunit.sourceforge.net/> - testare unitară în C++

VBUnit - <http://www.vbunit.org/> - testare unitară în Visual Basic

DUnit - <http://sourceforge.net/projects/dunit/> - testare unitară în Delphi

## **Referințe**

JUnit - site oficial: <http://www.junit.org>

JUnit Primer: <http://www.clarkware.com/articles/JUnitPrimer.html>

Ant - site oficial: <http://ant.apache.org/>

*Refactoring: Improving The Design Of Existing Code*, de Martin Fowler (Addison-Wesley, 1999)

## **Despre autor**

Sorin Scorțan este inginer software la firma SecureNet din Craiova și poate fi contactat prin e-mail la adresa [ss@secure-net.ro](mailto:ss@secure-net.ro)