LABORATOR 1

Mediul de programare NetBeans

Cuvinte cheie : Dezvoltarea de aplicații, Mediu integrat IDE, Proiect, Depanare, Refactorizare, Testare unitară

Dezvoltarea de aplicaţii Java-Dezvoltarea de aplicaţii (*Application Development*) este un termen care reuneşte totalitatea operaţiilor necesare pentru obţinerea şi menţinerea unui produs software comercial: scrierea iniţiala de module, legarea modulelor într-un program unitar, testarea şi depanarea produsului, actualizarea programului la modificarea unor cerinţe sau extinderea programului cu noi funcţii, arhivarea într-un singur fişier, instalarea programului pe calculatorul beneficiarului sau pe un server de aplicaţii (*Deployment*), etc.

Timpul necesar dezvoltării de aplicaţii mai mari poate fi mult redus prin utilizarea unui mediu integrat de dezvoltare (*IDE= Integrated Development Environment*), care asigură o serie de facilităţi pentru editarea programelor, pentru depanarea lor, pentru construirea de aplicaţii şi pentru testarea lor, toate cu o interfaţă grafică prietenoasă şi usor de utilizat. Un astfel de produs integrează mai multe programe diferite utilizate în linie de comandă: editor de texte, compilator, depanator, suport pentru teste unitare (JUnit), constructor de aplicaţii, server Web, etc.

Cele mai utilizate produse IDE pentru Java sunt *Eclipse*, *NetBeans* si *Intellij IDEA* care permit dezvoltarea de aplicaţii în mai multe limbaje: Java, C, C++, PHP, Groovy, ş.a. In plus, ele pot fi folosite şi ca medii vizuale, pentru crearea interfeţelor grafice ale aplicaţiilor

Mediul NetBeans poate fi descărcat în câteva variante: numai pentru aplicaţii Java standard (SE = Standard Edition), numai pentru aplicaţii C,C++, numai pentru aplicaţii PHP, pentru aplicaţii Web în Java (JEE = Java Enterprise Edition) sau pentru toate variantele anterioare plus alte facilităţi (Java ME, Groovy, Java Card).

Orice aplicaţie dezvoltată sub NetBeans necesită crearea unui proiect. Crearea unui nou proiect se face prin selectarea opţiunii *File* din meniul principal şi apoi a opţiunii *New Project* (din File) sau prin scurtătura Ctrl-Shift-N.

Aplicaţiile reale conţin multe clase şi alte fişiere. Beneficiarul aplicaţiei primeşte de obicei o arhiva de tip jar în care se împachetează toate fişierele necesare folosirii aplicaţiei. Crearea arhivei se face fie din meniu: Run > Clean and Build Main Project sau cu scurtătura Shift-F11. Arhiva este plasată în sub-directorul *dist* (pentru distribuţie) din directorul proiect.

Pentru includerea în proiect a unor biblioteci de clase Java (altele decât cele standard SDK): Extindere nod proiect > click–dreapta pe ―Libraries‖> Add JAR/Folder > selectare nume/cale fişier de tip jar sau *folder*.

Depanarea programelor (*Debugging*) se face prin selectarea opţiunii *Debug* din meniul principal sau prin Ctrl-F5. In acest mod se pot utiliza următoarele procedee: Stablilirea unor puncte de oprire temporară (*Breakpoints*), Execuţia pas-cu-pas (*Step-Over*) a instrucţiunilor sursă, Execuţia până la poziţia cursorului, Inspectarea conţinutului unor variabile din program, Modificarea valorii unor variabile din panoul de variabile.

Refactorizare în NetBeans - Refactorizarea codului sursă se referă la modificarea disciplinată a codului fără a modifica efectul codului la execuţie. Refactorizarea conduce la surse mai uşor de înţeles, de modificat (de întreţinut) şi permite găsirea şi eliminarea unor erori. Cea mai folosită operaţie de refactorizare este schimbarea numelui unui pachet sau unei clase, unei metode sau unei variabile. NetBeans modifică toate apariţiile acelui nume, inclusiv referiri la el. Pentru schimbarea unui nume avem două posibilităţi: clic dreapta pe nume > Refactor > Rename sau din meniu selectăm Refacor > Rename.

Exemple de alte operaţii de refactorizare:
- Inlocuirea unui bloc de cod cu o metodă prin introducerea definiţiei metodei şi apelului metodei (*Introduce Method*).
- Generarea metodelor *get()*şi *set()* pentru un câmp şi înlocuirea referinţelor la câmp prin apeluri de metode (*Encapsulate Fields*)
- Mutarea unei clase într-un alt pachet (*Move Class*)
- Ştergere sigură (Safely Delete), dacă nu mai sunt referinte la acel element.
- Modificarea parametrilor unei metode (Change Method Parameters) .

Testarea programelor Java în NetBeans-Testarea unitară este parte din procesul dezvoltării de programe, iar pentru aplicaţiile Java se foloseşte preponderent biblioteca de clase ***JUnit.*** In NetBeans se pot folosi atât versiunea 4 JUnit, cu adnotări, cât şi versiunea 3 fără adnotări.

Testarea unitară verifică, pentru fiecare unitate de program (metodă Java), faptul că, pentru anumite date iniţiale, rezultatele sunt cele aşteptate. In acest scop se scrie pentru fiecare metodă testată o funcţie de test în care se folosesc aserţiuni. Se poate folosi instrucţiunea ***assert*** din Java sau aserţiuni JUnit. O aserţiune este o afirmaţie care poate fi adevărată sau falsă; dacă este adevărată programul continuă, iar dacă este falsă atunci se produce excepţia *AssertionFailedError*.

Exemple de aserţiuni:

assertTrue (boolean condition) Trece dacă condiţie adevărată

assertEquals(Object expected,Object actual) Trece dacă obiectele sunt egale după metoda equals() assertEquals (int expected, int actual) Trece dacă cele două valori sunt egale (==). Valorile pot fi de orice tip primitiv Java.

assertSame(Object expected, Object actual) Trece dacă cele două obiecte sunt identice assertNull(Object object) Trece dacă argumentul este null

Fiecare metodă de tip assert poate avea un parametru suplimentar de tip *String* care este un mesaj afişat în caz că aserţiunea (condiţia) este falsă.

In funcţiile de test se folosesc în general obiecte ale clasei testate; aceste obiecte pot fi create în interiorul metodei sau în afara metodelor de test. In acest scop sunt prevăzute în clasa de test metodele cu numele ***setUp()*** si ***tearDown()*** (JUnit 3) sau metode cu orice nume dar adnotate cu ***@Before*** si ***@After***. Metoda ***setUp()*** creează obiectele necesare metodelor de test şi initializează variabile ale clasei care vor fi folosite în metodele de test, înainte de fiecare test. Metoda pereche ***tearDown()*** anulează operatiile din ***setUp()***şi reface starea iniţială, dacă e necesar.

Testele unitare pot fi clasificate în trei categorii:

- teste pozitive : se verifică rezultatul aşteptat al unei acţiuni
- teste negative: se verifică comportarea în cazul unor date de intrare (parametri) cu erori
- teste de producere excepţii: se verifică dacă sunt tratate excepţiile posibile

Exemplul folosit aici este o clasă cu două metode: înmulţire şi împărţire de întregi:

```
public class Calc {

public int mul (int a, int b) { return a*b; }

public int div (int a, int b) { return a/b; }

}
```

Exemplu de fişier cu codul de testare a clasei *Calc* scris manual folosind JUnit 3:

```
import org.junit.*;

import static org.junit.Assert.*;

import junit.framework.*;

public class UnitTest1 extends TestCase {

Calc c;

 public void setUp(){
```

```
c=new Calc();
}
public void testMul() {
assertEquals (c.mul(2,3),6) ; // assertTrue ( c.mul(2,3)==6);
}
public void testDiv() {
assertEquals (c.div(6,3),2) ; // assertTrue ( c.div(6,3)==2);
 }
}
class Runner {
public static void main (String a[ ]){
org.junit.runner.JUnitCore.main("UnitTest1");
}
 }
```

Pentru generarea de teste JUnit în NetBeans se face click dreapta pe numele clasei sau pe numele pachetului de clase (dacă sunt mai multe clase supuse testelor), se selectează

Tools > Create Tests > Framework > JUnit

Putem apoi alege între versiunile 3 şi 4 de JUnit. Efectul este acela de generare a unui nou pachet de clase în directorul *TestPackages* cu clase de test pentru fiecare clasă din aplicaţie. In cazul clasei *Calc* se generează clasa de test următoare după selecţia variantei JUnit 3:

```
import junit.framework.TestCase;
public class CalcTest extends TestCase {
public CalcTest(String testName) {
super(testName);
}
@Override
```

```
protected void setUp() throws Exception {
 super.setUp();
}
@Override
protected void tearDown() throws Exception {
super.tearDown();
 }
 public void testDiv() {
 System.out.println("div");
int a = 0;
 int b = 0;
 Calc instance = new Calc();
 int expResult = 0;
 int result = instance.div(a, b);
 assertEquals(expResult, result);
// TODO review the generated test code and remove the default call to fail.
fail("The test case is a prototype.");
 }
}
 . . . // testMul
```

Aşa cum indică şi comentariile vom elimina liniile finale care apelează metoda ***fail()*** si vom cere execuţia testelor în una din variantele:

- Din meniul principal: **Run > Test Project**

- Click dreapta pe numele proiectului > **Test (sau Alt-F6)**

Deoarece se generează automat date de test (a,b) cu valoarea zero se va produce o excepţie neprevăzută şi netratată în metoda ***div()***, iar testul eşuează (*Test failed*) din cauza acestei erori.

Pentru testarea metodelor care pot genera excepţii se poate verifica în metoda de test dacă excepţia a fost tratată sau nu în metoda verificată. Dacă adăugăm metodei *div* tratarea exceptiei aritmetice atunci testul va trece cu bine.

Exemplu:

```
public void testDiv() {
int a = 0;
int b = 0;
 int result=0;
 Calculator instance = new Calculator();
int expResult = 0;
try {
result = instance.div(a, b);
 }
catch (ArithmeticException ex){
fail ("ArithmeticException");
}
 assertEquals(expResult, result);
}
```

Teste generate în varianta JUnit 4:

```
public class CalcTest {
 public CalcTest() { }
@BeforeClass
public static void setUpClass() { }
@AfterClass
public static void tearDownClass() { }
```

```
@Before public void setUp() { }
@After
public void tearDown() { }
@Test
public void testMul() {
 System.out.println("mul");
int a = 0;
 int b = 0;
Calc instance = new Calc();
 int expResult = 0;
 int result = instance.mul(a, b);
assertEquals(expResult, result);
// TODO review the generated test code and remove the default call to fail.
fail("The test case is a prototype.");
}
@Test
 public void testDiv() {
 . . .
}
 }
```

Dacă metoda testată aruncă o excepţie (tratată în altă metodă din aplicaţie) atunci putem specifica în metoda de test că se asteaptă producerea acelei excepţii:

```
@Test (expected=ArithmeticException.class)
public void testDiv() { …}
```

Exerciții

1. Descărcați și instalați :
    a. JDK 8: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
    b. Netbeans 8.0.2 varianta JavaEE: https://netbeans.org/downloads/
2. Să se testeze exemplele de la tutorial-ul: https://netbeans.org/kb/docs/java/junit-intro.html
3. Să se testeze exemplele din acest laborator (de mai sus).
4. Să se scrie un program Java care utilizează o clasă *Suprafata* care să conțină metodele :
    a. *triunghi* cu parametrii *baza* și *inaltimea* – calculează aria triunghiului
    b. *dreptunghi* cu parametrii *lungime* și *latime* - calculează aria dreptunghiului
    c. *romb* cu parametrii *diagonala1* și *diagonala2* – calculează aria rombului
    d. *cerc* cu paramatrul *raza* – calculează aria cercului
5. Să se testeze metodele clasei *Suprafata* cu JUnit 4
6. Să se testeze exemplele de la tutorial-ul: http://tutorials.jenkov.com/java-unit-testing/simple-test.html