Sumar

1. Import JUnit în proiectul IntelliJ IDEA 2
2. Crearea unui Test Case folosind JUnit 3
3. Scrierea unui Test Case folosind JUnit 4
4. Execuția Test Case-urilor create cu JUnit 5
5. Localizarea bug-urilor. *Test Case* vs. *Tested Method* 6
6. *JUnit 3* vs. *JUnit 4* 6

Lista de Figuri

Figure 1. Deschiderea ferestrei de configurare a proiectului 2
Figure 2. Adăugarea dependenţelor la proiect 2
Figure 3. Adăugarea junit-4.12.jar 3
Figure 4. Adăugarea hamcrest-core 1.3.jar 3
Figure 5. Crearea unei clase pentru testarea entităţii Carte 4
Figure 6. Configurarea clasei de testare 4
Figure 7. Afişarea rezultatului execuţiei testelor 5

Lista de Tabele

Table 1 *JUnit 3* vs. *JUnit 4* 10

Tutorialul pentru utilizarea platformei de testare JUnit in IntelliJ IDEA poate conține anumiți pași care pot fi omiși.

1. Import JUnit în proiectul IntelliJ IDEA

1. în **IntelliJ**, click dreapta pe *numele proiectului* în **Project Explorer** ---> **Open Module Settings** (vezi Figure 1)**;**
2. în tab-ul **Dependencies**, click pe **+** şi se alege opțiunea **Add JARs or directories...** (vezi Figure 2)**;**
3. se selectează folder-ul în care este instalat IntelliJ (e.g., **C:\Program Files\JetBrrains\IntelliJIDEA 2019.2**)**;**
4. se deschide folder-ul **lib** şi se selectează fişierul **junit-4.12.jar** ---> **OK** (vezi Figure 3)**;**
5. similar, din folder-ul **lib** adauga la proiect fişierul **hamcrest-core-1.3.jar** (vezi Figure 4)**;**
6. click pe **OK** pentru salvarea dependențelor adăugate la proiect.

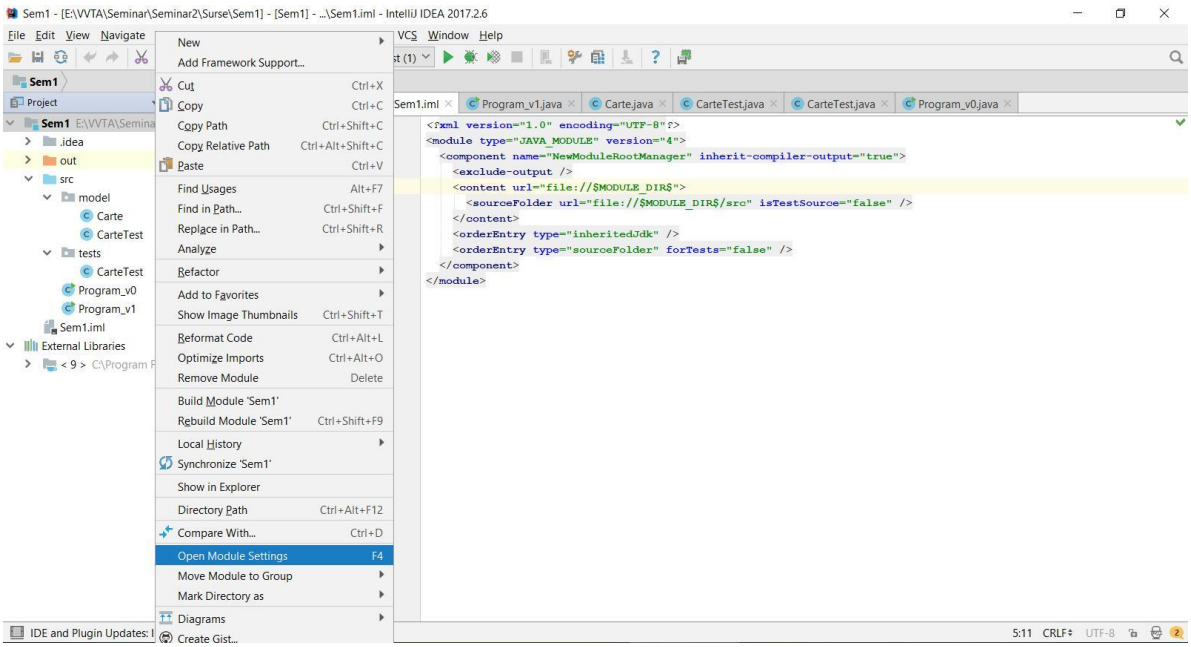

Figure 1. Deschiderea ferestrei de configurare a proiectului

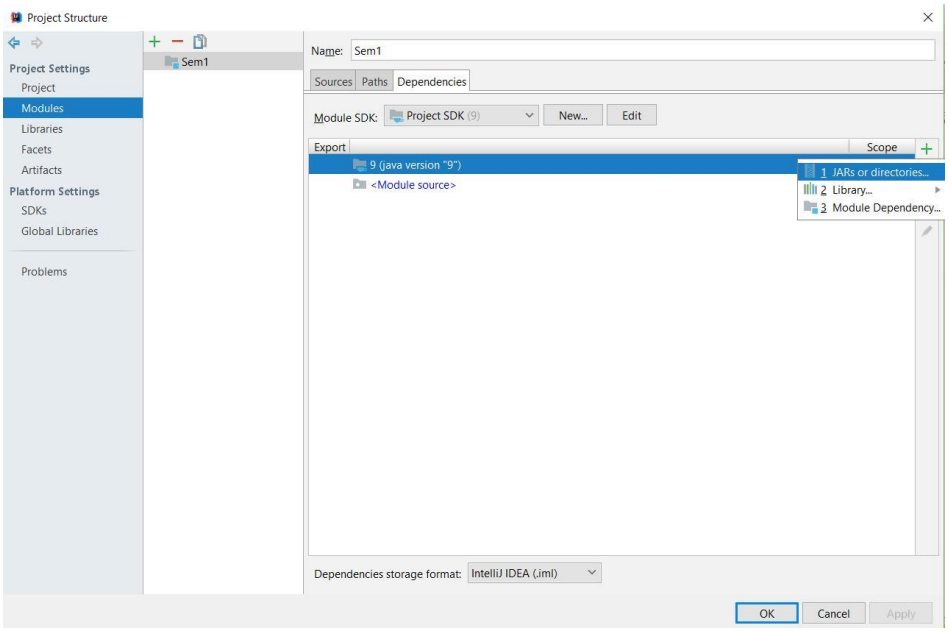

Figure 2. Adăugarea dependenţelor la proiect

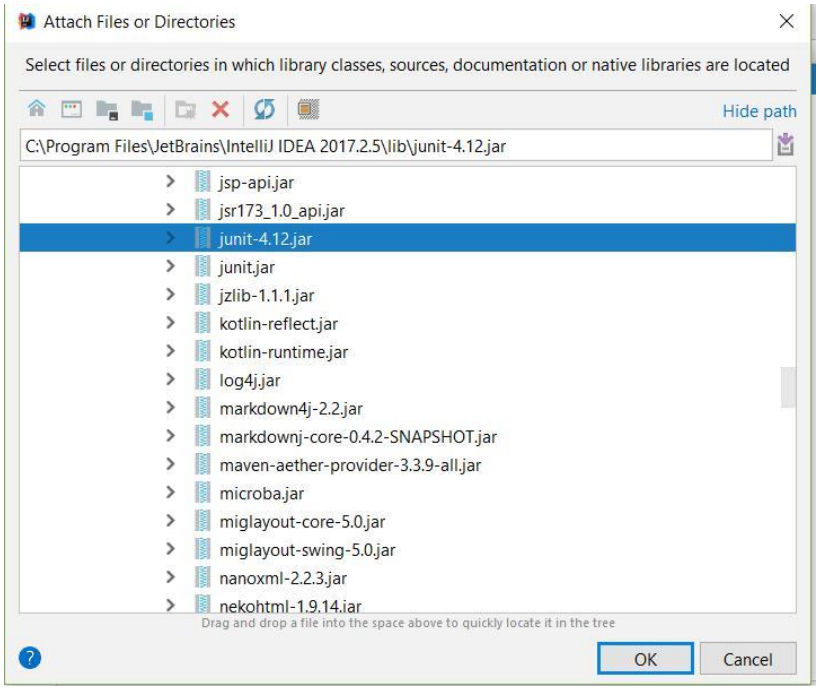

Figure 3. Adăugarea junit-4.12.jar

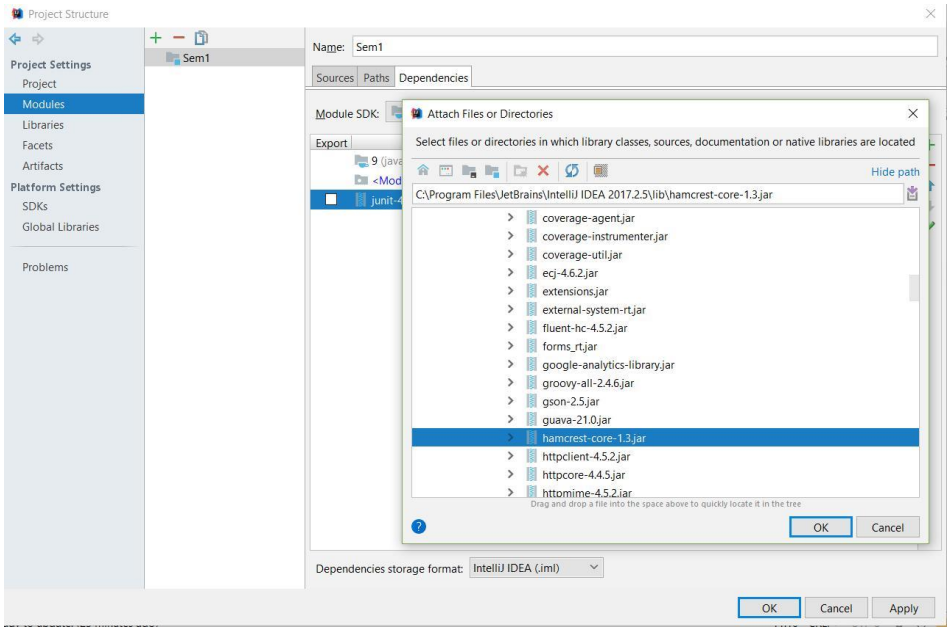

Figure 4. Adăugarea hamcrest-core 1.3.jar

2. Crearea unui Test Case folosind JUnit

1. în **IntelliJ,** se poziționează cursorul pe numele clasei;
2. se folosește combinația de taste **Alt+Enter**;
3. din meniul pop-up care apare se selectează opțiunea **Create Test** (vezi Figure 5);
4. se va deschide o fereastră care permite (vezi Figure 6):
   - alegerea platformei de testare: **JUnit 3.x** sau **JUnit 4.x**;
   - stabilirea numelui clasei care va conține teste (**Class name**);

- (opțional) se poate bifa utilizarea metodelor stub:
  - `setUp()` – pentru inițializarea stării înainte de execuția fiecărui test;
  - `tearDown()` – pentru finalizarea testului (e.g., revenirea la starea anterioară execuției testului);
  - **cele două metode se apelează implicit înainte şi după fiecare test rulat;**
- (opțional) se pot alege metodele din clasa testate pentru care să se creeze teste stub (i.e., metode care vor fi descrise ulterior);
- dacă se alege **JUnit 4.x**, testele stub generate vor fi adnotate cu **@Test**;
- clickOK pentru crearea clasei cu teste.

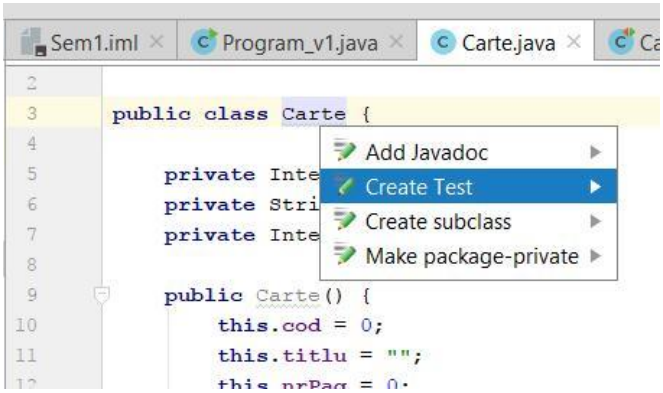

Figure 5. Crearea unei clase pentru testarea entității Carte

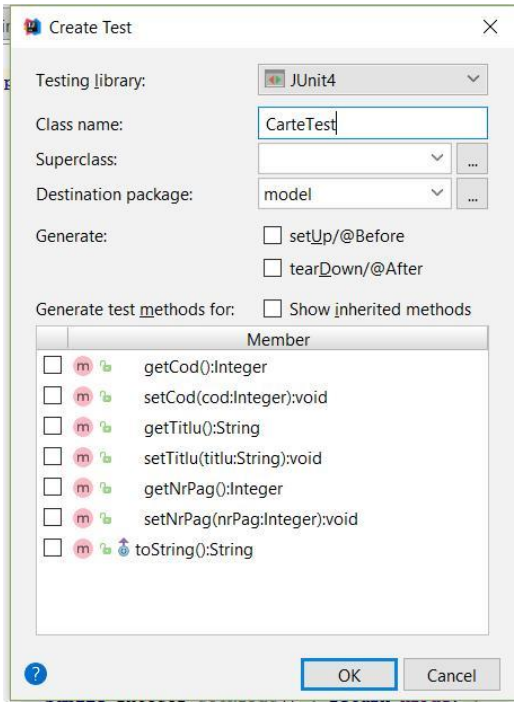

Figure 6. Configurarea clasei de testare

3. Scrierea unui Test Case folosind JUnit

1. **Exemplu** 1: testarea metodei `getTitlu()`:
   - se declară două atribute `private` de tip `Carte` în clasa `CarteTest`:

```
private Carte c1;
private Carte c2;
```

- referințele se inițializează în metoda `setUp()`:
```
c1 = new Carte(1,"Povesti", 198);
c2 = new Carte(2,"Povestiri",170);
```
- înainte de execuția fiecărei metode de testare se execută metoda `setUp()` care va instanția cele două obiecte de tip `CarteTest`; adică la rularea fiecărui **Test Case** vor exista obiectele de tip `Carte` `c1` și `c2`;
- în **Test Case**-ul `testGetTitlu()` se adaugă codul de testare pentru metoda `getTitlu()`:
```
assertEquals("Povesti", c1.getTitlu());
```
- dacă în cadrul unui **Test Case** unul dintre apelurile metodei `assertAAA(...)` eşuează, execuția **Test Case**-ului se încheie imediat, iar JUnit setează statusul acestuia ca failed;

2. **Exemplu** 2: testarea metodei `simplify()`:
   - în **Test Case**-ul `testConstructor()` se adaugă codul de testare pentru testarea construirii unui obiect de tip `Carte`:
```
@Test
public void testConstructor(){
    Carte c3 = new Carte(10, "povestiri", 123);
    //assertNull(c3 == null);
    assertEquals(c3, null);
}
```

4. Execuția Test Case-urilor create cu JUnit

1. în **IntelliJ**, având clasa `CarteTest` ca şi clasă curentă, din meniul **Run** ---> **Run 'CarteTest' sau**
2. click dreapta pe numele clasei `CarteTest` în **Project Explorer** şi se alege opțiunea **Run 'CarteTest';**
3. *Java Perspective* se modifică prin apariția tab-ului **JUnit** sub **Project Explorer** (vezi Figure 7);

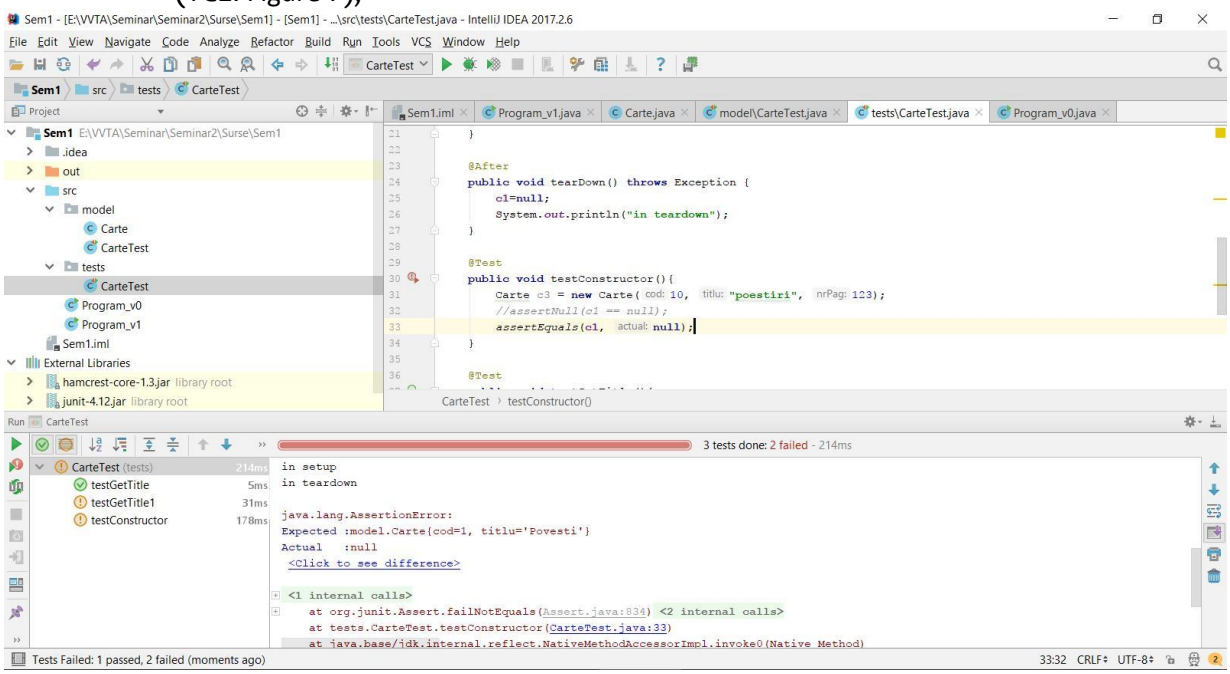

Figure 7. Afişarea rezultatului execuţiei testelor

4. terminarea cu succes a rulării **Test Case**-urilor duce la apariția barei de culoare verde în tab-ul **JUnit**, altfel aceasta are culoare roşie (vezi Figure 7);

5. pentru fiecare **Test Case** selectat din tab-ul teste executate, în frame-ul din partea dreaptă sunt oferite detalii cu privire la execuție (i.e., valoare așteptată, valoare obținută, excepții aruncate, etc.).

5. Localizarea bug-urilor. *Test Case* vs. *Tested Method*

1. bug-urile puse în evidență de execuția eșuată a unui **Test Case** sunt determinate de erori care pot apărea în:
   - codul sursă testat (e.g., `getTite()`) – **[tested_method#bug]**;
   - **Test Case**-ul propriu-zis (e.g., `testGetTitle()`) – **[test_case#bug]**;
2. se verifică dacă datele de intrare din etapa de proiectare a **Test Case**-ului au fost preluate corect în implementare;
   - după verificare și re-execuția **Test Case**-ului:
     - success ---> pentru datele de intrare furnizate, metoda obține rezultatele așteptate **[test_case#bug: fixed]**;
     - failed ---> există un bug în metoda testată **[tested_method#bug: needs debugging]**.

6. *JUnit 3 vs. JUnit 4*

1. Junit 4 beneficiază de introducerea adnotărilor Java (engl. ***annotation***);
2. **Test Case**-urile se pot scrie mult mai ușor, deoarece nu mai există restricții cu privire la alegerea identificatorilor metodelor;
3. în tabelul de mai jos sunt prezentate comparativ abordărilor existente în Junit 3.x și Junit4.x;
   - clasa `FractionClassTest_old` este o clasă cu teste care folosește platforma **Junit 3.x**;
   - clasa `FractionClassTest` este o clasă cu teste care folosește platforma **Junit 4.x**;

JUnit 3	JUnit 4	Observații
Crearea clasei **Test Case**		
`public class FractionClassTest_old extends TestCase { …}`	`public class FractionClassTest { …}`	
- orice clasă de testare este derivată din `TestCase`	- fără constrângeri referitoare la derivare	
Adnotarea **`@BeforeClass`**		
`public static void setUpBeforeClass() throws Exception { System.out.println("Setup for all subsequent tests..."); //setup }`	`@BeforeClass public static void setUpAll() { System.out.println("Setup for all subsequent tests..."); //setup }`	metodă statică ce se va executa o singură dată, înainte de rularea primului **Test Case** din clasă (e.g., conectarea la baza de date);
- metoda este denumită (obligatoriu) `setUpBeforeClass`;	- metoda este precedată de adnotarea `@BeforeClass`; - fără constrângeri referitoare la stabilirea identificatorului metodei;	
Adnotarea **`@AfterClass`**		
`public static void tearDownAfterClass() throws Exception { System.out.println("\ntearing all down"); }`	`@AfterClass public static void tearDownAll() { System.out.println("\ntearing all down"); }`	metodă statică ce se va executa o singură dată, după rularea tuturor **Test Case**-urilor din clasă (e.g., deconectarea de la baza de date);
- metoda este denumită (obligatoriu) `tearDownAfterClass`;	- metoda este precedată de adnotarea `@AfterClass`; - fără constrângeri referitoare la stabilirea identificatorului metodei;	
Adnotarea **`@Before`**		
`public void setUp() { fc1 = new FractionClass(12,30); fc2 = new FractionClass(-25,7); }`	`@Before public void setup() { fc1 = new FractionClass(12,30); fc2 = new FractionClass(-25,7); }`	metodă care se va executa înainte de fiecare **Test Case** din clasă (e.g., inițializarea cu date de intrare a testului);
- metoda este denumită (obligatoriu) `setUp`;	- metoda este precedată de adnotarea `@Before`; - fără constrângeri referitoare la stabilirea identificatorului metodei;	

<table>
<tr><th colspan="3">Adnotarea @After</th></tr>
<tr><td>

```
public void tearDown() {
	fc1 = fc2 = null;
	System.out.println(fc1);
	System.out.println(fc2);
}
```

</td><td>

```
@After
public void teardown() {
	fc1 = fc2 = null;
	System.out.println(fc1);
	System.out.println(fc2);
}
```

</td><td rowspan="2">metodă care se execută după fiecare Test Case din clasă (e.g., ştergerea/ dealocarea resurselor folosite de test);</td></tr>
<tr><td>- metoda este denumită (obligatoriu) <code>tearDown</code>;</td><td>- metoda este precedată de adnotarea <code>@After</code>;
- fără constrângeri referitoare la stabilirea identificatorului metodei;</td></tr>
<tr><th colspan="3">Adnotarea @Test</th></tr>
<tr><td>

```
public void testSimplify() {
	System.out.println("\ntestSimplify");
	fc1.Simplify();
	assertEquals(2, fc1.getNumerator());
	assertEquals(5, fc1.getDenominator());
}
```

</td><td>

```
@Test
public void mySimplifyTest() {
	System.out.println("\ntestSimplify");
	fc1.Simplify();
	assertEquals(2, fc1.getNumerator());
	assertEquals(5, fc1.getDenominator());
}
```

</td><td rowspan="2">metodă Test Case propriu-zisă (e.g., testează comportamentul funcției <code>simplify()</code> pentru anumite date de intrare care, de exemplu, au fost inițializate într-o metodă <code>setUp()</code> sau <code>setup()</code>);</td></tr>
<tr><td>- metoda începe (obligatoriu) cu prefixul <code>test</code> (e.g., <code>testSimplify</code>, <code>testGetDenominator</code>), altfel nu este considerată un Test Case;</td><td>- metoda este precedată de adnotarea <code>@Test</code>;
- fără constrângeri referitoare la stabilirea identificatorului metodei;</td></tr>
<tr><th colspan="3">Adnotarea @Test (expected = <<ClassException>>.class)</th></tr>
<tr><td>

```
public void testDivisionException(){
	System.out.println("\ntestDivisionException"
);
	fc2.setDenominator(0);
	try {
		fc1.div(fc2);
	} catch (Exception e) {
		e.printStackTrace();
	assertTrue(e.getMessage().equals("Division
by zero!"));
	}
}
```

</td><td>

```
@Test (expected = Exception.class)
public void testDivisionException() throws
Exception //passed {
	System.out.println("\ntestDivisionExceptio
n");
	fc2.setDenominator(0);
	fc1.div(fc2);
}
SAU
@Test(expected=IndexOutOfBoundsException.class)
public void outOfBounds()//passed {
	new ArrayList<Object>().get(1);
}
```

</td><td>metodă Test Case propriu-zisă care pune în evidență aruncarea unei excepții</td></tr>
</table>

<table>
<tr><td>- metoda începe (obligatoriu) cu prefixul <code>test</code> ;
- se folosește un apel assertAAA(…) care pune în evidență succesul sau eșecul testului;</td><td>- metoda este precedată de adnotarea <code>@Test</code> cu clauza <code>expected</code>, prin care se precizează tipul excepției așteptate la execuție;
- dacă metoda testată aruncă excepție, Test Case-ul va avea starea passed;
- fără constrângeri referitoare la stabilirea identificatorului metodei;</td><td></td></tr>
<tr><td colspan="3">Adnotarea <code>@Test (timeout = <<value>>)</code></td></tr>
<tr><td></td><td><pre>@Test (timeout=10)//passed
public void testDivision() {
	System.out.println("\ntestDivision");
	try {
		fc1.div(fc2);
	} catch (Exception e) {
		e.printStackTrace();
	}
	assertEquals(-14, fc1.getNumerator());
	assertEquals(125, fc1.getDenominator());
}
SAU
@Test(timeout=100)//failed
public void infinity() {
	while(true);
}</pre></td><td rowspan="2">metodă Test Case propriu-zisă care pune în evidență execuția într-un interval de timp precizat</td></tr>
<tr><td></td><td>- metoda este precedată de adnotarea <code>@Test</code> cu clauza <code>timeout</code>, prin care se precizează timpul maxim de execuție așteptat, exprimat în milisecunde;
- dacă la execuție timpul depășește valoarea dată, Test Case-ul va avea starea failed;
- fără constrângeri referitoare la stabilirea identificatorului metodei;</td></tr>
<tr><td colspan="3">Adnotarea <code>@Ignore</code></td></tr>
<tr><td><pre>public void ttestGetDenominator() {
	System.out.println("\ntestGetDenominator");
	int result = fc1.getDenominator();
	assertTrue("getDenominator() returned " +
result + " instead of 30.", result == 30);</pre></td><td><pre>@Ignore
@Test
public void testGetDenominator() {
	System.out.println("\ntestGetDenominator")
;</pre></td><td>metodă Test Case care va fi ignorată la rularea testelor (e.g.,</td></tr>
</table>

`result = fc2.getDenominator();`
`assertEquals(7, result);`
`}`
	`int result = fc1.getDenominator();`
	`assertTrue("getDenominator() returned " + result + " instead of 30.", result == 30);`
	`result = fc2.getDenominator();`
	`assertEquals(7, result);`
	`}`
		se foloseşte atunci când codul sursă se modifică, iar **Test Case**-ul corespunzător nu s-a adaptat încă).

- orice metodă care nu are prefixul `test` (e.g., `ttestGetDenominator()`) nu va fi considerată un **Test Case** şi va fi ignorată.	- metoda este precedată de adnotarea `@Ignore`; - fără constrângeri referitoare la stabilirea identificatorului metodei.	

Table 1 *JUnit 3* vs. *JUnit 4*