Programació I - Introducció bàsica als Tests Unitaris amb JUnit

UNIVERSITAT DE BARCELONA

Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica

5 de novembre de 2018

UNIVERSITAT DE BARCELONA

Continguts

Què és testejar?

- Testejar és el procés de comprovar la funcionalitat d'un programa per assegurar-se que **funciona segons els requisits** establerts.
- Fins ara hem fet "Manual Testing", amb una taula de tests de prova. Manualment comprovarem que el resultat esperat era igual al obtingut.
- Ara farem "Automatic Testing", concretament Unit testing (test unitaris - testeig d'un sol mòdul de codi). El procés serà molt similar, comparem resultat esperat i obtingut. Utilitzarem JUnit.

Què és JUnit?

- JUnit és un marc de proves unitàries per al llenguatge de programació Java (`http://JUnit.org/JUnit5/`).
- Pertany a l'arquitectura xUnit per a marcs de test unitaris.
- JUnit és una eina essencial en el desenvolupament basat en proves.
- JUnit facilita els tests de regressió (després de canvis en el codi i/o integració de nou codi, comprovar automàticament que tot segueix funcionant.)

Concepte JUnit: Cas de prova

- Un cas de prova (Unit Test Case) és una part de codi, que assegura que una altra part del codi (un mètode) funciona com s'esperava.
- Un cas de prova es caracteritza per una **entrada coneguda** i un **resultat esperat**.

Definició dels Casos de prova (Test Cases)

Els casos de prova es defineixen creant una nova classe:

- S'anomena `xxxTest`, on xxx es el nom de la classe on està el mètode a testejar (A JUnit 4.0 ja no és necessari però nosaltres ho farem igualment).
- Aquesta classe ha d'incloure els mètodes de prova amb el nom testxxx, on xxx és el nom del mètode a testejar (A JUnit 4.0 ja no és necessari però nosaltres ho farem igualment).

Definició dels Casos de prova (Test Cases)

Amb JUnit volem fer tests positius i negatius.

Diem que un **test és positiu** si estem donant dades de entrada valides. Testegem si el programa es comporta com s'espera amb aquests dades.

- Per exemple, un programa diu si un nombre entrat per teclat i amb un valor entre 1 i 100, és múltiple de 3. Si introduïm el nombre 50, es un test positiu i la sortida esperada serà «No és múltiple».

Tipus de test: positiu i negatiu

Diem que un **test és negatiu** si estem donant dades de entrada que no són vàlids.

- Per exemple, Un programa diu si un nombre, entrat per teclat i amb un valor entre 1 i 100, és múltiple de 3. Un test negatiu comprova la sortida pel nombre 104. La sortida pot ser un missatge dient «el valor no està entre 1 i 100».

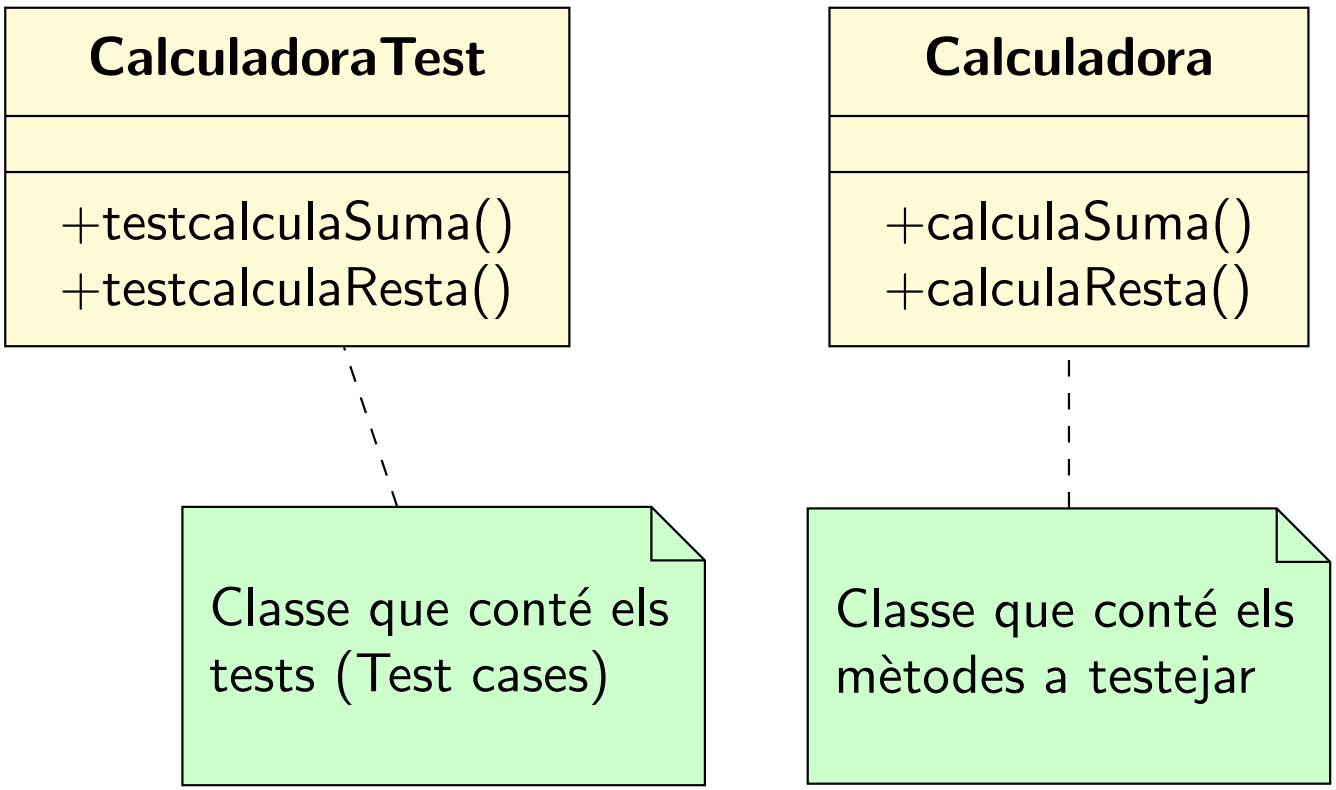
Procediment per a fer un testeig

Passos:

1. Definim la classe que fa el testeig (Test cases).
2. Definim la classe amb els mètodes que volem testejar.
3. Compilem i executem.

NOTA: En un desenvolupament basat en proves (Test-driven development-TDD), fem pas 1 i pas 2 en aquest ordre, això vol dir que el desenvolupador escriu els Tests Cases abans d'escriure el codi. En un desenvolupament més clàssic es fa a l'ordre invers.

Definició dels Casos de prova (Test Cases) i definició de la Classe a testejar



JUnit a Netbeans

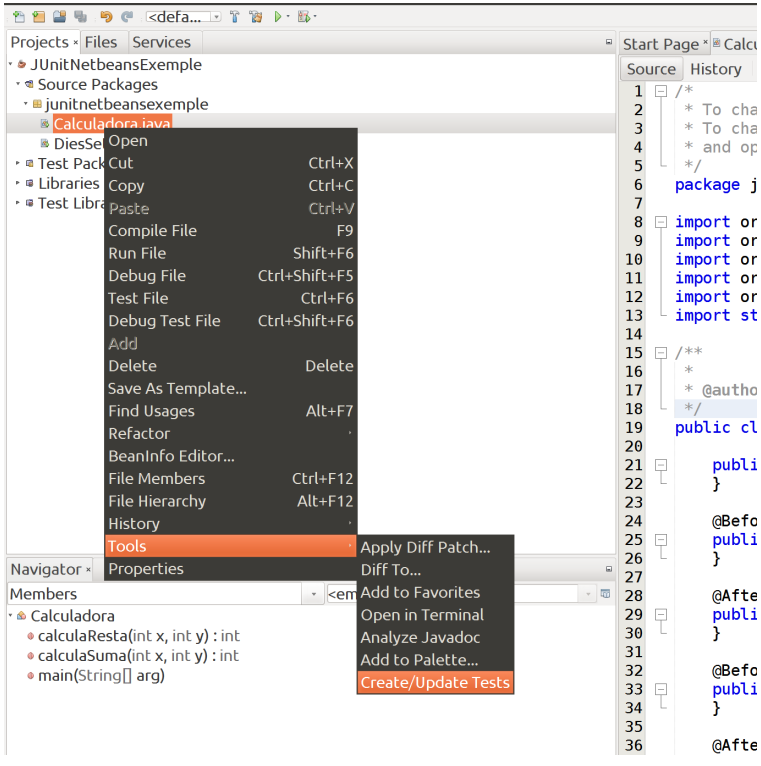
Netbeans té integrat el suport del framework JUnit i permet fer test de manera ràpida i sencilla.

Per tal de fer ús de les utilitats integrades de JUnit necessiteu tenir instal·lat i activat el plugin de JUnit per a Netbeans. Podeu instal·lar dit plugin a través del Plugins manager de Netbeans que podeu trobar al menú Tools.

A continuació veurem com podem crear tests de les nostres classes del projecte i com executar-los i veure'n el resultat.

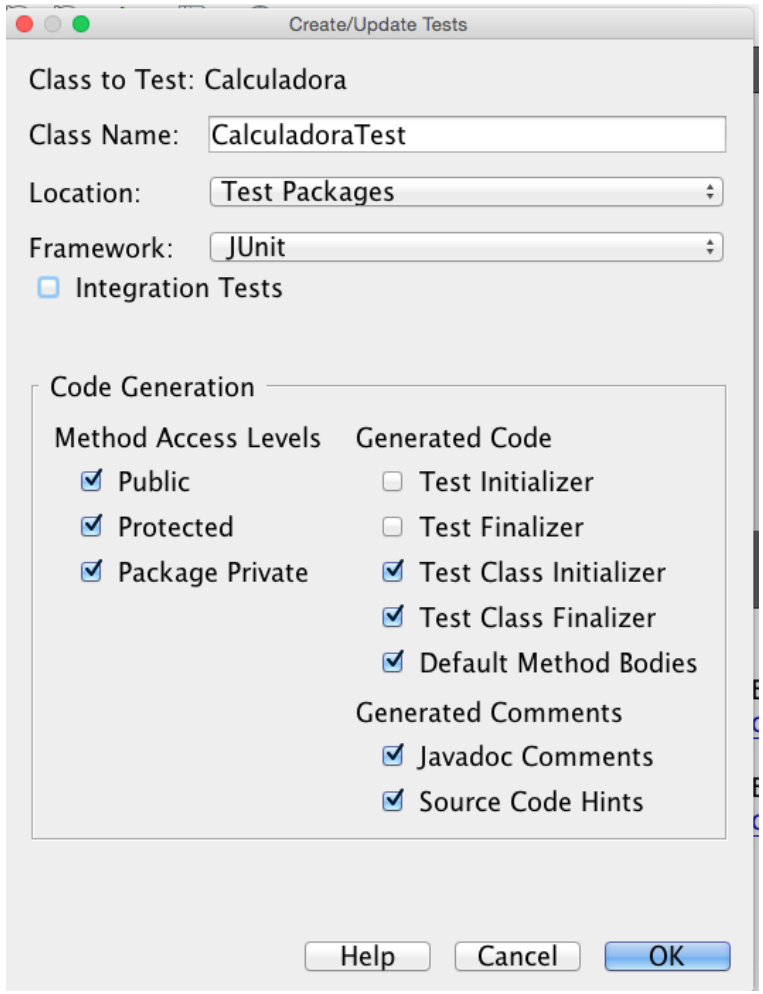
Creant les classes de Test a partir de les classes del projecte

- Pas 1: Baixeu el projecte **CalculadoraAlumne** del CV. Obriu-lo.
- Pas 2: Per tal de crear les classes de test només cal que feu click dret a l'arxiu .java (en el nostre cas `Calculadora.java`) del qual volem crear el test i seleccioneu Tools → Create Tests.
  Si us pregunten per la versió de JUnit escolliu 4.x.



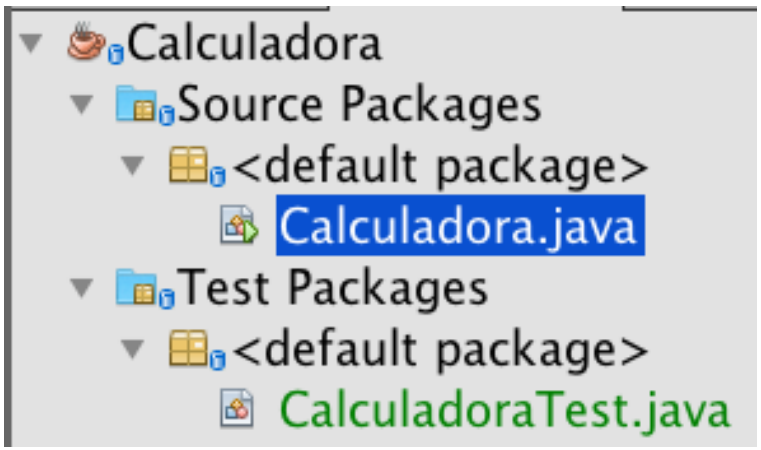
Creant les classes de Test a partir de les classes del projecte

S'obrirà un diàleg amb opcions de generació de codi, deixeu les opcions per defecte. Feu clic a OK.



Creant les classes de Test a partir de les classes del projecte

L'IDE crearà automàticament la classe `CalculadoraTest.java` a un package amb el mateix nom que el package contenidor de Calculadora.java però sota el node de Test Packages a la finestra de projecte. Feu una ullada a la classe generada al vostre projecte.



Modificant els mètodes de test generats per l'IDE

- Pas 3: Ara ja podeu modificar cadascun dels mètodes generats.

Primer, poseu entre comentaris el codi automàticament generat per a testejar el mètode `main()`. Només volem testejar els mètodes `calculaSuma()` i `calculaResta()`.

```
1  /*@Test
2      public void testMain() {
3          System.out.println("main");
4          String[] arg = null;
5          Calculadora.main(arg);
6          //TODO review the generated test code and remove the
               default call to fail.
7          fail("The test case is a prototype.");
8      }
9      */
```

Modificant els mètodes de test generats per l'IDE

Segon, modifiqueu `testCalculaSuma()` com s'indica a la figura a sota.

```
1  /*Test of calculaSuma method, of class Calculadora.*/
2  @Test
3      public void testCalculaSuma() {
4          System.out.println("calculaSuma");
5          int x = 3;
6          int y = 4;
7          int expResult = 7;
8          int result = Calculadora.calculaSuma(x, y);
9          assertEquals(expResult, result);
10         // TODO review the generated test code and remove the
               default call to fail.
11         //fail("The test case is a prototype.");
12     }
```

`assertEquals()`, això què vol dir? → Assegura't que són iguals!!

- Si la crida a `Calculadora.calculaSuma (x,y)` retorna en `result` el mateix valor que el primer paràmetre del `assertEquals (expResult)`, el sistema ens dirà que el **Test és vàlid**.
- En cas contrari (si no són iguals expResult i `result`) ens dirà que el **Test no és vàlid**.

Modificant els mètodes test generats per l'IDE

Tercer, modifiqueu `testCalculaResta()` com s'indica a la figura a sota.

```
1  /*Test of calculaResta method, of class Calculadora.*/
2      @Test
3      public void testCalculaResta() {
4          System.out.println("calculaResta");
5          int x = 1;
6          int y = 4;
7          int expResult = -3;
8          int result = Calculadora.calculaResta(x, y);
9          assertEquals(expResult, result);
10         // TODO review the generated test code and remove the
               default call to fail.
11         //fail("The test case is a prototype.");
12     }
```

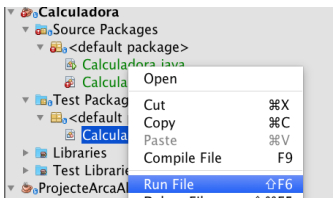
`assertEquals()`, això què vol dir? → Assegura't que són iguals!!

- Si la crida a `Calculadora.calculaResta (x,y)` retorna en `result` el mateix valor que el primer paràmetre del `assertEquals (expResult)`, el sistema ens dirà que el **Test és vàlid**.
- En cas contrari (si no són iguals expResult i `result`) ens dirà que el **Test no és vàlid**.

Executant els Tests

- Pas 4: Executar el test. Podeu executar tots els tests d'una aplicació sencera o els tests individuals per a cada classe.
  - Per executar tots els tests unitaris del projecte aneu al **menú principal Run** → **Test** <**nom-projecte**>. L'IDE executarà totes les classes de test al package de Test Packages.

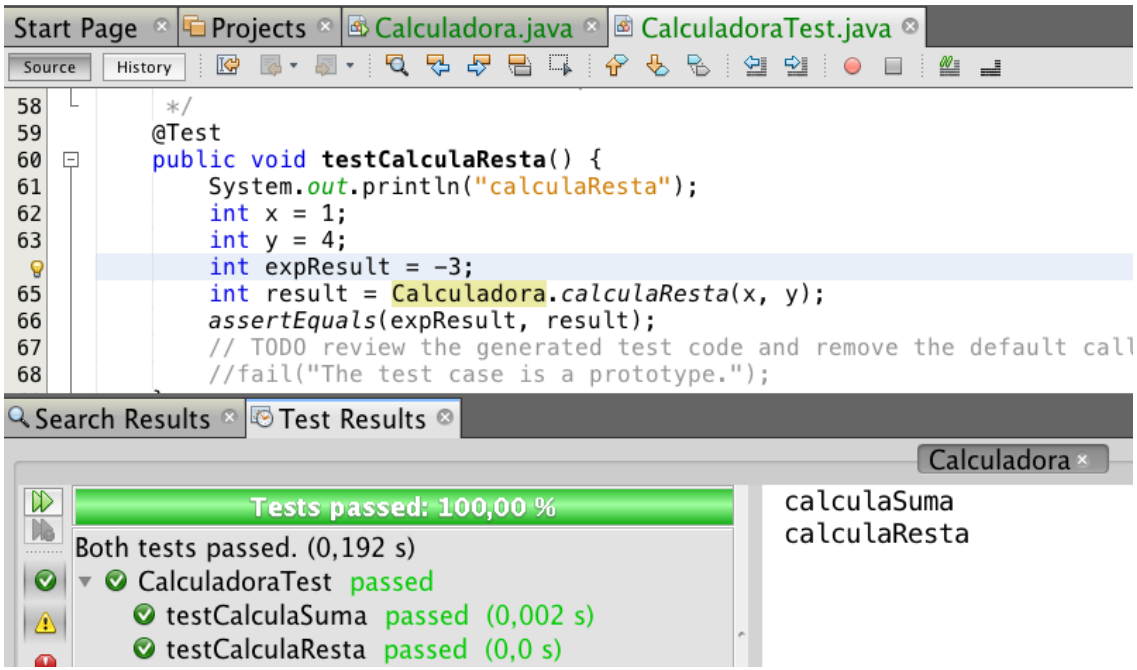
  - Per executar un test individual d'una classe només cal fer **clic dret a la classe de test que vulgueu de Test Packages i fer Run** File.



Executant els Tests

Finestra de resultats de test

Si no veieu aquesta finestra de resultats podeu fer Window → IDE Tools → Test Results. Si tots els tests es passen apareixerà quelcom semblant a:

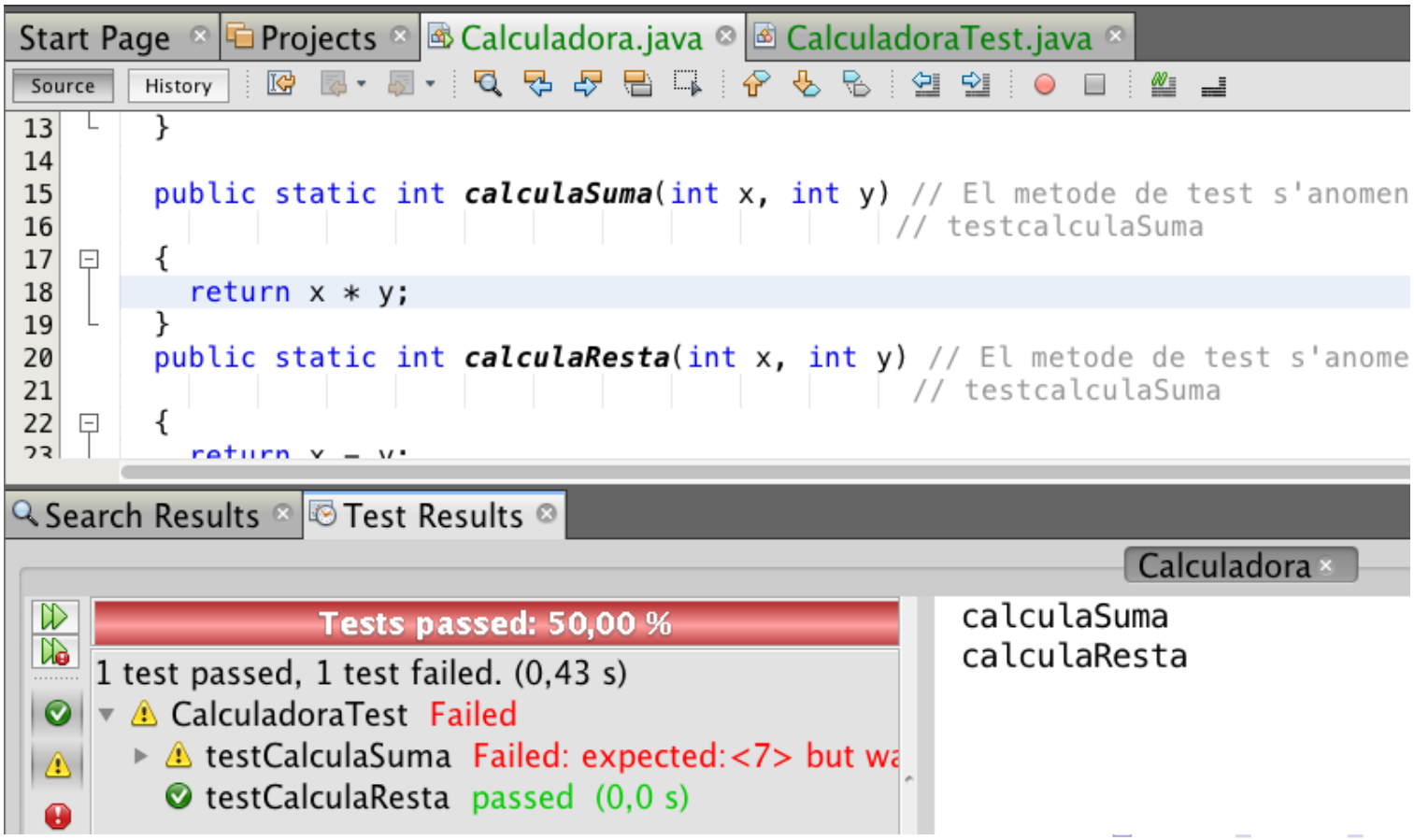


El panell esquerra mostra els resultats dels mètodes individuals del test i el panell dret la sortida del test.

Executant els Tests

Finestra de resultats de test

Si en canvi, algun test ha fallat podrem veure en detall els tests que han fallat i el mètode que ha provocat la fallada. Fixeu-vos al `x * y` del mètode `calculaSuma()`.



Mètodes assert

Alguns mètodes de la classe Assert de junit:

- `assertTrue(boolean)`, testeja que una condició sigui true.
- `assertFalse(boolean)`, testeja que una condició sigui falsa.
- `assertEquals(Object, Object)`, testeja que dos objectes siguin iguals.
- `assertNull(Object)`, testeja que un objecte sigui null.
- `assertNotNull(Object)`, testeja que un objecte no sigui null.
- `assertThat(T actual, org.hamcrest.Matcher<T> matcher)`, testeja que actual satisfà la condició especificada per matcher.
- ...

Mètodes assert: exemples

- `assertTrue(boolean)`, testeja que una condició sigui true.

```
1  x = 2, y=0;
2  assertTrue(x > y);  //passa el testeig
3  assertTrue(x < y);  //no el passa
```

- `assertNotNull(Object)`, testeja que un objecte no sigui null.

```
1  String s = new String("Prova");
2  String s2 = null;
3  assertNotNull(s);//passa el testeig
4  assertNull(s);    //no el passa
5  assertNull(s2); //passa el testeig
```

- `assertThat(T actual, org.hamcrest.Matcher<T> matcher)`, testeja que actual satisfà la condició especificada per matcher.

```
1  assertThat(0, is(1));  // no passa el
       testeig
2  assertThat(0, is(not(1)));//passa el
```

Més sobre JUnit...suite tests

- Un suite test és un conjunt de tests de prova.
- JUnit executa suite tests i reporta resultats.
- Per a cada test de prova en el suite tests:
- JUnit crida `setUp()`, per a crear variables o objectes que necessites pel testing.
- JUnit crida `tearDown()`, per eliminar aquestes variables i objectes, quan ja no els necessites.
- JUnit crida un mètode de test (anomenat testxxx(),essent xxxx el nom del mètode que vols provar).
- Nosaltres (per ara) no farem suit test.

Junit a la consola

A continuació s'explica com fer els tests de prova a la consola.

1. Classe que fa el testeig: CalculadoraTest

```
1  import org.JUnit.Assert;
2  import org.JUnit.Test;
3
4  public class CalculadoraTest{
5     @Test
6     public  void  testcalculaSuma( ){
7       Calculadora c = new Calculadora();
8
9       int s = c.calculaSuma(3,4);
10      assertEquals(7, s);
11    }
12 }
```

El mètode **assertEquals** és un mètode de JUnit que comprova que els dos arguments siguin iguals. Veurem altres mètodes més endavant.

2. Classe amb els mètodes a testejar: Calculadora

```
1  import java.util.Scanner;
2  public class Calculadora{ //CalculadoraTest
3    public static void main(String [ ] arg){
4      int a, b, c;
5      Scanner sc = new Scanner(System.in);
6      System.out.println("Nombres enters a sumar: ");
7      a = sc.nextInt();
8      b = sc.nextInt();
9      c = calculaSuma(a,b);
10     System.out.println(c);
11   }
12   public static int calculaSuma(int x, int y)
13   {
14     return x + y;
15   }
16 }
```

3. Compilem i executem (en ordre els tres comandes a continuació)

```
1 $javac␣Calculadora.java
2 $javac␣-cp␣.:junit-4.10.jar␣
     CalculadoraTest.java
3 $java␣-cp␣
     .:junit-4.10.jar:hamcrest-core-1.3.jar␣
     org.junit.runner.JUnitCore␣CalculadoraTest
```

Recordeu: en el mateix directori dels Calculadora*.java, hem de tenir: junit-4.10.jar i hamcrest-core.jar Baixeu-los de:

`https://github.com/JUnit-team/JUnit4/wiki/Download-and-Install` `http://central.maven.org/maven2/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.jar`

4. Resultat (Passa el test positiu)

```
Inma-Mac:Junit inma$ java -cp .:junit-4.10.jar:hamcrest-core-1.3.jar org.
nitCore CalculadoraTest
JUnit version 4.10
.
Time: 0.046

OK (1 test)
```

5. Resultat (No passa el test positiu)

Si introduïm un error a la implementació del mètode `calculaSuma()`:

```
1 public static int calculaSuma(int x, int y){
2     return x * y;
3 }
```

Si tornem a compilar Calculadora.java, en executar el test obtenim:

```
Inma-Mac:Junit inma$ java -cp .:junit-4.10.jar:hamcrest-core-1.3.jar
org.junit.runner.JUnitCore CalculadoraTest
JUnit version 4.10
.E
Time: 0.006
There was 1 failure:
1) testcalculaSuma(CalculadoraTest)
java.lang.AssertionError: expected:<7> but was:<12>
```

Bibliografia

JUnit:
`http://JUnit.org/JUnit4/` `http://JUnit.org/JUnit4/faq.html`
JUnit a netbeans:
https://netbeans.org/kb/docs/java/JUnit-intro.html