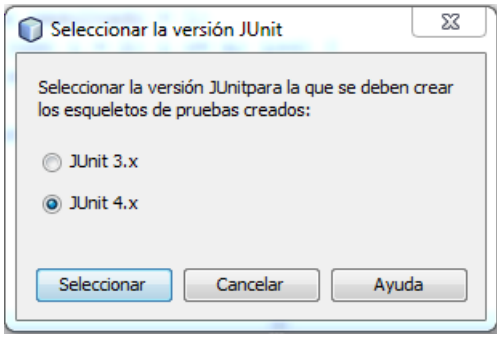
Probas unitarias con Junit

JUnit é un framework ou conxunto de clases Java que permiten ao programador construír e executar automaticamente casos de proba para métodos dunha clase Java. Os casos de proba quedan reflectidos en programas Java que quedan arquivados e poden volver a executarse tantas veces como sexa necesario. Estes casos de proba permiten avaliar se a clase se comporta como se esperaba, é dicir, a partir duns valores de entrada, avalíase se o resultado obtido é o esperado. JUnit foi escrito por Erich Gamma e Kent Beck e é un produto de código aberto distribuído baixo unha licenza Common Public License - v 1.0. A páxina oficial de JUnit é : www.junit.org.
Os IDE como NetBeans, JBuilder e Eclipse contan con complementos para utilizar JUnit, permitindo que o programador se centre na proba e no resultado esperado e deixe ao IDE a creación das clases que permiten a proba.
Neste documento utilizarase o plugin coas versións 3 e 4 das librerías de probas unitarias JUnit para o IDE NetBeans 8.0.1 e faranse as probas deste apartado sobre o sinxelo proxecto *proyecto_division* composto das clases Division.java e Main.java seguintes:

Xerar probas en JUnit

En NetBeans pódense xerar tres tipos de probas: para probar os métodos dunha clase, un conxunto de probas para probar un conxunto de clases dun paquete ou un caso de probas JUnit baleiro.
Para crear unha proba JUnit, pódese seleccionar *Archivo Nuevo* na opción *Archivo* do menú principal, seleccionar a categoría *Unit Tests* e o tipo de arquivo de entre os tres posibles: *JUnit Test* que crea un caso de proba baleiro, *Test for Existing Class* que crea un caso de proba para os métodos dunha clase e *Test Suite* que crea probas para un paquete Java seleccionado. Dependendo do tipo de proba seleccionada, haberá que completar de forma diferente as seguintes ventás que aparecen.
Outra maneira de poder crear as probas de clase é seleccionar a clase ou package sobre o que se queren facer as probas e elixir no menú principal *Herramientas->Create/Update Tests*.
Se non se creou anteriormente ningunha proba para ese proxecto, aparece a ventá *Seleccionar la versión JUnit* para permitir elixir entre traballar coa versión 3 ou 4. Deixase seleccionada a versión 4 xa que esta inclúe á versión 3.



En calquera caso aparece unha ventá diferente se a proba é para unha clase ou para un paquete. Nos apartados seguintes só se contemplarán os casos de proba para unha clase e para un package.

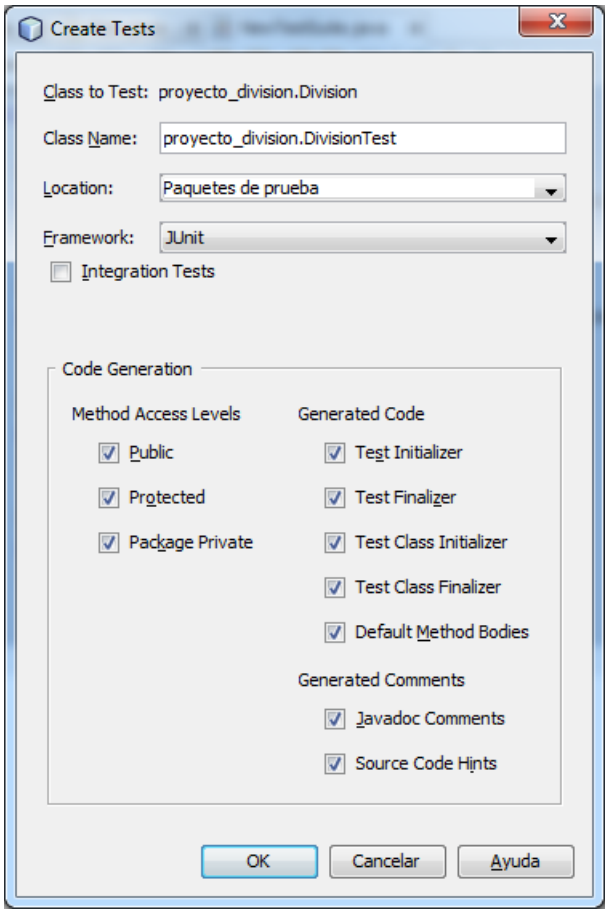
Para unha clase

De elixir no menú principal *Herramientas->Create/Update Tests*, aparecerá a ventá *Create Tests* na que hai que indicar: o nome da clase de proba (recoméndase deixar o nome por defecto: nome da clase a probar seguido de *Test*), a localización onde se gardará a proba (recoméndase deixar o valor por defecto) e as características da xeración de código dividida en tres apartados:

- Apartado *Method Access Levels* para indicar o nivel de acceso aos métodos de proba.

- Apartado *Generated Code* para indicar se a proba terá métodos para executarse antes de iniciar a proba (*Test Initializer*), despois de finalizar a proba (*Test Finalizer*) ou código exemplo para unha proba (*Default Method Bodies*).
- Apartado *Generated Comments* para indicar que as probas leven comentarios Javadoc e comentarios para suxerir como implementar os métodos de proba (*Source Code Hints).*

Déixase todo seleccionado e prémese en OK.



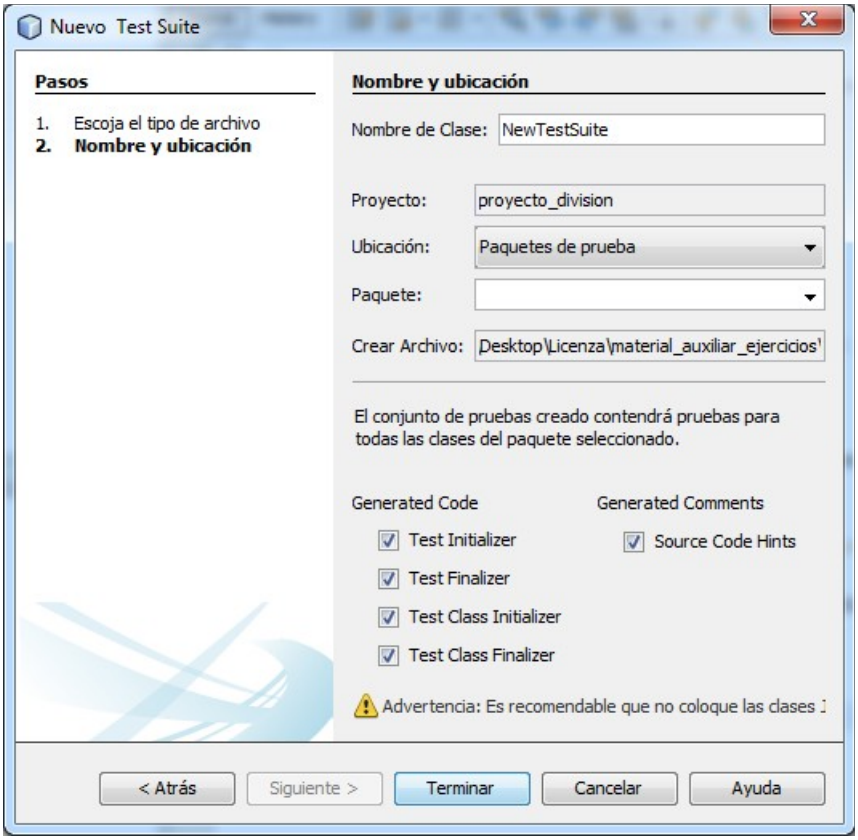
O código que xera NetBeans por defecto para a proba do método *calcularDivision()* contempla un só caso de proba (0/0=0). Pódese modificar este método e engadir novos casos de proba ou engadir outros métodos de proba. NetBeans suxire borrar as dúas últimas liñas de código da anotación *Test*. Pódense borrar ou comentar as liñas das anotacións agás a anotación *Test* e as liñas *import* correspondentes ás anotacións borradas, se é que non van a ser utilizadas.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package proyecto_division;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Profesor
 */
public class DivisionTest {

    public DivisionTest() {
    }
```

```
    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of calcularDivision method, of class Division.
     */
    @Test
    public void testCalcularDivision() throws Exception {
        System.out.println("calcularDivision");
        float dividendo = 0.0F;
        float divisor = 0.0F;
        Division instance = new Division();
        float expResult = 0.0F;
        float result = instance.calcularDivision(dividendo, divisor);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

}
```

Para un package

De elixir no menú principal Archivo->*Archivo Nuevo*->*Unit Tests*->*Test Suite*, aparecerá a ventá *New Test Suite*, onde se dará nome á clase de proba e elixirase o paquete a probar, premendo ao final en *Finish*.



Na ventá de edición pódese ver o código que por defecto xera NetBeans para a proba. Na liña 20

pódese ver que as clases de proba que xa existían (como neste exemplo a clase *DivisionTest*) están incluídas no conxunto de probas. Pódense engadir novas probas.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package proyecto_division;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

/**
 *
 * @author Profesor
 */
@RunWith(Suite.class)
@Suite.SuiteClasses({proyecto_division.DivisionTest.class})
public class NewTestSuite {

    @BeforeClass
    public static void setUpClass() throws Exception {
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

}
```

Estrutura da proba

As anotacións e métodos que aparecen por defecto son:

- A anotación *@BeforeClass* marca ao método *setUpClass()* para ser executado antes de empezar a proba de clase, é dicir, execútase unha soa vez e antes de empezar a execución dos métodos de proba. Pódese utilizar por exemplo para crear unha conexión cunha base de datos.
- A anotación *@AfterClass* marca ao método *tearDownClass()* para ser executado ao finalizar a proba de clase. Pódese utilizar por exemplo para pechar a conexión coa base de datos realizada antes.
- A anotación *@Before* marca ao método *setUp()* para ser executado antes da execución de cada un dos métodos de proba, é dicir, execútase tantas veces como métodos de proba existan. Utilízase para inicializar recursos, variables de clase ou atributos que sexan iguais para tódalas probas.
- A anotación *@After* marca ao método *tearDown()* para executarse xusto despois da execución de cada un dos métodos de proba.
- A anotación *@Test* marca cada método de proba.
- O método *assertEquals*. Este método afirma que o primeiro argumento (resultado esperado) é igual ao segundo (resultado obtido). Se os dous argumentos son reais, pode ter un terceiro argumento chamado valor delta, que é un número real igual á máxima

diferenza en valor absoluto entre o valor esperado e o actual para que a afirmación sexa un éxito.

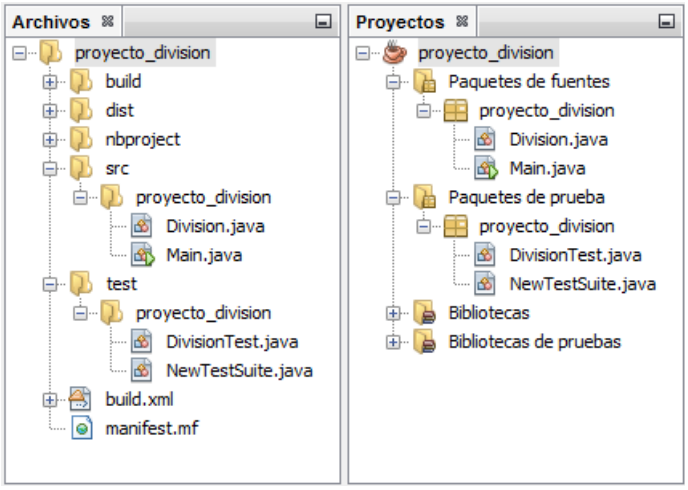
```
Math.abs(esperado-obtido)<delta
```

Na proba para o método *calcularDivision()* utilizada de exemplo, pódese modificar o test para poder comprobar que a división entre 1 e 3 dá como resultado 0.33 cun valor delta de 1E-2. Se consideramos que o método *calcularDivision(1,3)* devolve 0.333, os valores esperados 0.34 e 0.33 serían equiparables ao valor real e o valor 0.32 non, xa que abs(0.333-0.33) < 0.01, abs(0.333-0.34) < 0.01 e abs(0.333-0.32) > 0.01. O código do test podería ser:

```
package proyecto_division;

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

public class DivisionTest {

    @Test
    public void testCalcularDivision() throws Exception {
        System.out.println("Caso: 1/3=0.33 con valor delta 1E-2");
        Division instance = new Division();
        float resultado = instance.calcularDivision(1F,3F);
        assertEquals(0.33, resultado, 1E-2);

    }
}
```

Carpeta para as probas

Nas ventás de arquivos e de proxectos poden verse a estrutura lóxica e física das carpetas nas que se gardan as probas JUnit. Pódense agregar outras carpetas de proba no cadro de diálogo de propiedades do proxecto pero tendo en conta que os arquivos de proba e os fontes non poden estar na mesma carpeta.



Executar a proba

Pasos para executar a proba dun proxecto enteiro:

- Selecciónase calquera nodo ou arquivo do proxecto na ventá de proxectos ou na de arquivos e elíxese no menú principal *Ejecutar->Probar Project(nome_do_proxecto)* ou prémese Alt-F6.
- O IDE executa tódolos métodos de proba do proxecto. De querer executar un subconxunto das probas do proxecto ou executar as probas nun orden específico, debe crearse unha *Test Suite* que especifique as probas a executar.

Para executar a proba dunha clase, pódese elixir unha das dúas posibilidades seguintes:

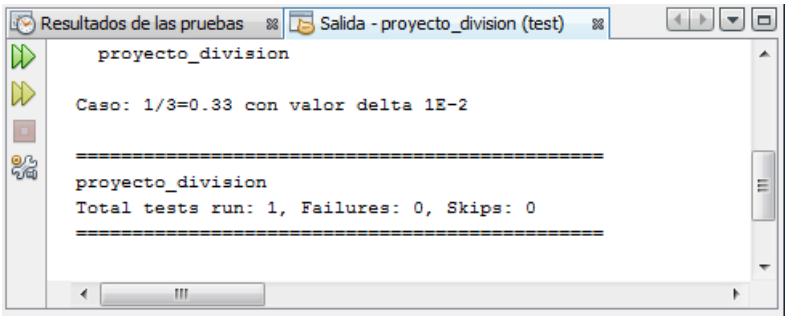
- Selecciónase a clase na ventá de proxectos ou na de arquivos, clic dereito e elíxese *Probar archivo* ou prémese Ctrl-F6.
- Selecciónase a proba da clase e elíxese no menú principal *Ejecutar -> Ejecutar archivo* ou prémese Mayús-F6.

Pasos para executar un caso de proba:

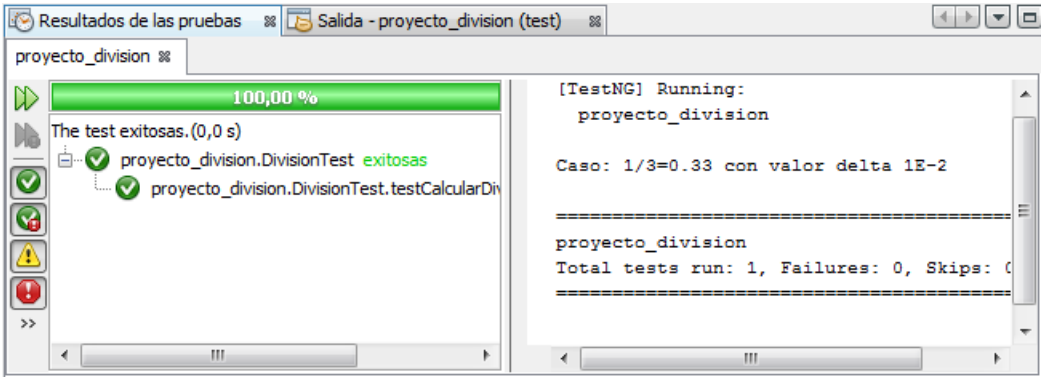
- Execútase a proba que contén ese caso de proba.
- Na ventá de resultados faise clic dereito sobre o método e elíxese *Run Again*.

Ventás de saída e resultados

A ventá de saída reflicte o detalle do proceso de execución das probas en formato texto.



A ventá de resultados ten dúas zonas. A zona da esquerda contén un resumo dos casos de proba superados e non superados e unha descrición deles en formato gráfico. Ao pasar o rato por riba da descrición dun método de proba, aparece nun recadro a saída correspondente a ese método. A zona da dereita contén a saída textual.



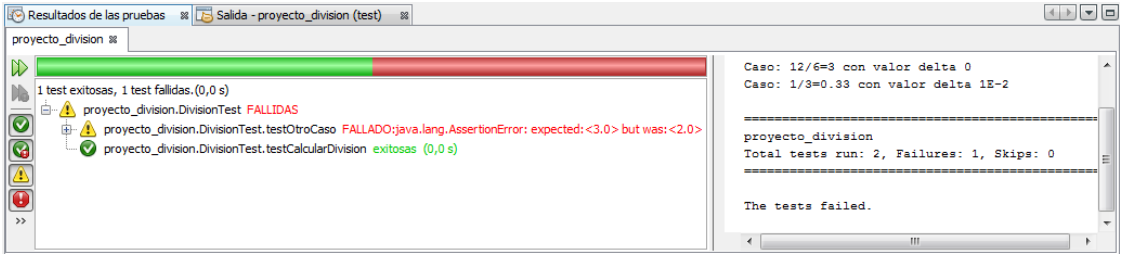
Algunhas das iconas que se poden utilizar na ventá de resultados son:

volver a executar a proba
volver a executar as probas non superadas
mostrar en detalle as probas superadas
mostrar en detalle as probas non superadas
mostrar en detalle os erros das probas non superadas
moverse polos métodos non superados cara arriba ou cara abaixo

Estas ventás cambian se non se supera algunha proba. Por exemplo, ao executar un método de proba que espera un resultado 3 para o caso de proba 12/6 como o seguinte:

```
@Test
public void testOtroCaso() throws Exception {
    System.out.println("Caso: 12/6=3 con valor delta 0");
    float dividendo = 12.0F;
    float divisor = 6.0F;
    Division instance = new Division();
    float expResult = 3F;
    float result = instance.calcularDivision(dividendo, divisor);
    assertEquals(expResult, result, 0.00);
}
```

Aparecerá na ventá de resultados o detalle sobre o caso de proba non superado.



Asertos

A clase *Assert* permite comprobar se a saída do método que se está probando concorda cos valores esperados. Pódese ver a sintaxe completa dos métodos asertos (afirmacións) que se poden usar para as probas en http://www.junit.org/apidocs/org/junit/Assert.html. Unha breve descrición deles é:
void assertEquals(valor esperado, valor actual) é un método con sobrecarga para cada tipo en java e que permite comprobar se a chamada a un método devolve un valor esperado. No caso de valores reais ten un terceiro argumento para indicar o valor delta ou número real igual á máxima diferenza en valor absoluto entre o valor esperado e o actual para que a afirmación sexa un éxito.
void fail() para cando se espera que o programa falle. Utilízase cando a proba indica que hai un erro ou cando se espera que o método que se está probando chame a unha excepción.
void assertTrue(boolean) a proba é un éxito se a expresión booleana é certa.
void assertFalse(boolean) a proba é un éxito se a expresión booleana é falsa.
void assertNull(Object) a proba é un éxito se o obxecto é nulo.
void assertNotNull(Object) a proba é un éxito se o obxecto non é nulo.
void assertSame(Object,Object) a proba é un éxito se os dous obxectos son o mesmo.
void assertNotSame(Object,Object) a proba é un éxito se os dous obxectos non son o mesmo.

Anotacións

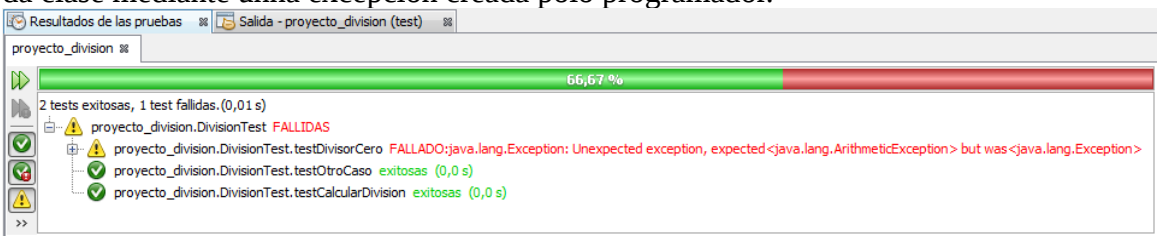
As anotacións aportan información sobre un programa e poden ser usadas polo compilador (por exemplo: *@Override*-para informarlle que se está sobrescribindo un método, *@Deprecated* para indicarlle que está en desuso, *@SupressWarnings* para indicarlle que non avise de *warnings* ou advertencias), por ferramentas de software que as poden procesar (por exemplo para xerar código ou arquivos xml) ou para ser procesadas en tempo de execución. As anotacións poden aplicarse a declaracións de clases, campos, métodos e outros elementos dun programa.
As anotacións máis importantes nunha proba son *@ Ignore* que serve para desactivar un test e colócase xusto antes de *@Test*, e *@Test*.
A anotación *@Test* serve para anotar unha proba. Pode ter dous parámetros opcionais *expected* e *timeout*. O primeiro define a clase de excepción que se espera que lance a proba para que sexa superada e o segundo define os milisegundos que como máximo debe durar a execución para que a proba sexa superada.
Exemplo con *timeout*: o seguinte test non se superará xa que a execución da proba supera os 100 milisegundos:

```
@Test(timeout = 100)
public void infinity() {
        while (true);
}
```

Exemplo con *expected*: se na proba da clase *Division* se engade un novo caso de proba para a división entre 0 contemplando unha excepción de tipo aritmético.

```
@Test(expected = ArithmeticException.class)
public void testDivisorCero() throws Exception {
        System.out.println("Caso: Divisor 0");
        Division instance = new Division();
        float resultado = instance.calcularDivision(10F, 0F);
}
```

Isto provocará que a proba non sea exitosa xa que a división por cero está contemplada no código da clase mediante unha excepción creada polo programador.



Unha posible solución sería contemplar unha excepción *java.lang.Exception*.

```
@Test(expected = java.lang.Exception.class)
public void testDivisorCero() throws Exception {
        System.out.println("Caso: Divisor 0");
        Division instance = new Division();
        float resultado = instance.calcularDivision(10F, 0F);
}
```