2- Probas unitarias con JUNIT

Táboa de contidos

1.Probas unitarias con JUNIT

Ata o de agora, temos realizado probas de forma manual. Neste epígrafe veremos como funciona un programa que realiza probas para verificar o noso programa.

A ferramenta que utilizaremos para as probas automatizadas será **JUnit**. Estará integrada en Eclipse polo que non deberemos descargarnos ningún paquete. Estas probas realizaranse sobre unha clase independentemente do resto de clases.

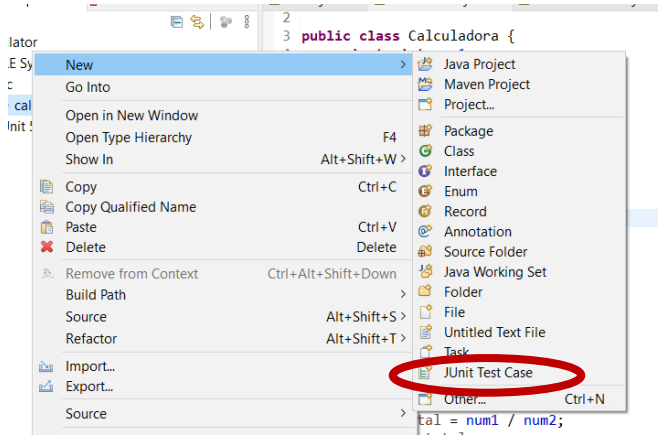
1.1. Creación da clase de proba

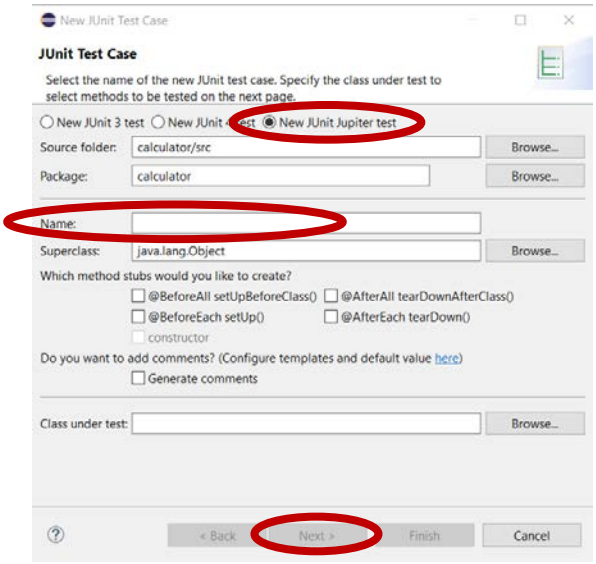
Crear un novo proxecto en Eclipse. Crearemos unha calculadora con 4 operacións: suma, resta, multiplica e divide.

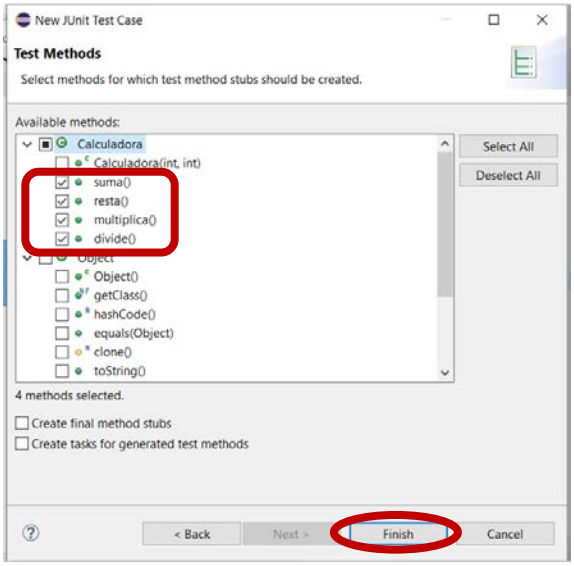
```
public class Calculadora {
	private int num1;
	private int num2;

	public Calculadora(int a, int b) {
		num1 = a;
		num2 = b;
	}
	public int suma() {
		int total = num1 + num2;
		return total;
	}
	public int resta() {
		int total = num1 - num2;
		return total;
	}
	public int multiplica() {
		int total = num1 * num2;
		return total;
	}
	public int divide() {
		int total = num1 / num2;
		return total;
	}
}
```

1.2. Creación da clase de proba

Pulsando co botón dereito enriba da clase, seleccionar **New**, e no despregable clicar sobre **JUnit Test Case**.







As clases de proba créanse automaticamente:

```
class CalculadoraTest {

	@Test
	void testSuma() {
		fail("Not yet implemented");
	}

	@Test
	void testResta() {
		fail("Not yet implemented");
	}

	@Test
	void testMultiplica() {
		fail("Not yet implemented");
	}

	@Test
	void testDivide() {
		fail("Not yet implemented");
	}

}
```

2. Preparación e execución das probas

Existen varios tipos de métodos (que devolven *void*) para realiza comprobacións:

MÉTODOS	MISIÓN
assertTrue(boolean expresión) **assertTrue(String mensaxe, boolean expression)**	Comproba que a expresión avalíase **true**. Se non é **true** e inclúese o *String*, ao producirse erro lanzarase a mensaxe
assertFalse(Boolean expresión) **assertFalse(String mensaxe, Boolean expresión)**	Comproba que a expresión se avalíase **false**. Se non é **false** e inclúese o *String*, ao producirse erro lanzarase a mensaxe.
assertEquals(valorAgardado, valorReal) **assertEquals(String mensaxe, valorAgardado, valorReal)**	Comproba que o valorAgardado sexa igual ao valorReal. Se non son iguais e inclúese o *String*, entón lanzarase a mensaxe. ValorAgardado e ValorReal poden ser de diferentes tipos.
assertNull(Object obxecto), **assertNull(String mensaxe, Object obxecto)**	Comproba que o obxecto sexa **null**. Se non é **null** e inclúese o *String*, ao producirse erro lanzarase a mensaxe.
assertNotNull(Object obxecto), assertNotNull(String mensaxe, Object obxecto)	Comproba que o obxecto non sexa **null**. Se non é **null** e inclúese o *String*, ao producirse erro lanzarase a mensaxe.
assertSame(Object obxectoAgardado, Object obxectoReal) **assertSame(String mensaxe, Object obxectoAgardado, Object obxectoReal)**	Comproba que **obxectoAgardado** e **obxectoReal** sexan o mesmo obxecto. Se non son o mesmo e inclúese o *String*, ao producirse o erro lanzarase a mensaxe.
assertNotSame(Object obxectoEsperad, Object obxectoReal) **assertNotSame(String** mensaxe**,Object obxectoAgardado, Object obxectoReal)**	Comproba que **obxectoAgardado** e **obxectoReal** non sexan o mesmo obxecto. Se non son o mesmo e inclúese o *String*, ao producirse o erro lanzarase a mensaxe.
fail() **fail(String mensaxe):**	Fai que a proba falle. Se se inclúe un *String* a proba falla lanzando a mensaxe.

Os casos de proba compóñense de 3 partes

1) Defínese unha variable co valor esperado
2) Execútanse o método a probar cos datos de entrada adecuados e gárdase o resultado nunha variable
3) Comprobase a relación entre o valor esperado e o resultado obtido a través dunha das aserción anteriores. Neste caso a aserción é **assertequals()**.

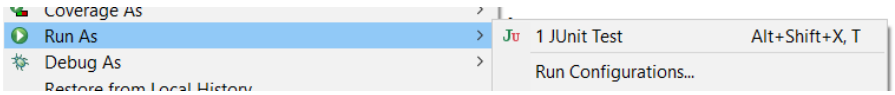
O seguinte é un exemplo de código para o método **testSuma()** que probará o método **suma()** da clase a probar.

```
@Test
void testSuma() {
	double valorEsperado = 30;
	Calculadora calcu = new Calculadora(20, 10);
	double resultado = calcu.suma();
	assertEquals(valorEsperado, resultado, 0);
}
```

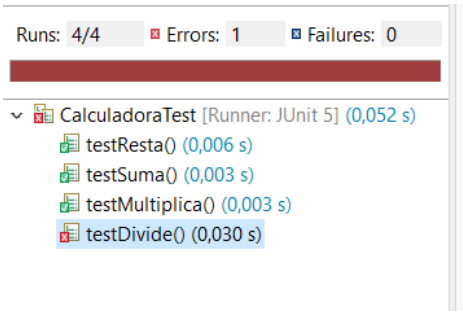
Fai ti os outros métodos!!!!

2.1. Executar a clase de proba

Para executar a clase de proba premer sobre a clase test co botón dereito e seleccionar **Run as > JUnit Test**.



E obterase o resultado da proba



3. Tipos de anotacións

JUnit ten dispoñible unhas anotacións que permite executar o código **antes** e **despois** das probas:

- @BeforeEach: se anotamos un método con esta etiqueta, o código será **executado antes de calquera método de proba**. Pódese empregar para iniciar datos, por exemplo, nunha aplicación de acceso a base de datos, inicializar un array, pode haber varios métodos nunha clase con esta anotación.
- @AfterEach: se anotamos un método con esta etiqueta, o código será **executado despois da execución de cada método de proba**. Pódese empregar para limpar datos. Pode haber varios métodos nunha clase con esta anotación.
- @BeforeAll: só poderá ter esta etiqueta un só método. É invocado unha vez ao principio do lanzamento de todas as probas. Útil para inicializar atributos comúns a todas as probas ou para realizar accións que tardan un tempo considerable en executarse.
- @AfterAll: só poderá ter esta etiqueta un só método. Será invocado unha soa vez cando finalicen todas as probas.

4. Probas parametrizadas

Para executar unha proba varias veces, pero con distintos valores.

Decláranse utilizando a anotación @ParameterizedTest en lugar de **@test**. Deberase declarar alo menos unha fonte que proporcionará os argumentos para cada invocación e despois consumir os argumentos no método de proba.

@ValueSource permite especificar unha única matriz de valores literais e so pode usarse para proporcionar un único argumento ao método de proba.

No exemplo realizaranse dúas probas do método, unha co valor **Ola** e outra co valor **Mundo.**

```
@ParameterizedTest
@ValueSource(String = {“Ola”, “Mundo”})
void mensajeNoNulo(String cadena) {
      assertNotNull(cadena);
}
```

Para especificar valores enteiros empregaremos `@ValueSource(ints = {10, 20, 9})`
Para proporciona a un método de proba varios parámetros temos que utilizar a anotación @CsvSource, configurando os datos de proba usando unha matriz de obxectos String, tendo en conta:

- O obxecto String debe conter todos os parámetros para unha invocación do método.
- Os diferentes parámetros deben separarse cunha coma.
- Os valores deben seguir a mesma orde que os parámetros definidos no método de proba

```
@ParameterizedTest
@CsvSource( {“10, 20, 9”,
             “30, -2 -15”,
             “8, 2, 3” })
public void testDivide1 (int a, int b, int valorEsperado) {
      Calculadora calcu = new Calculadora(a, b);
      int resultado = calcu.divide();
      assertEquals(valorEsperado, resultado);
}
```

5. Suite de probas

Ás veces é interesante executar varias clases de proba unha tras outra. Para elo **JUnit** proporciona o mecanismo chamado **Test Suites** que permite agrupar varias clases de proba para que se executen unha a continuación da outra. Na suite de probas pódense agruparas

clases que forman parte dun paquete, ou ben pódense indicar os nomes das clases que formarán parte da suite.

Para crear a suite de probas temos que crear unha clase con algunhas das seguintes anotacións:

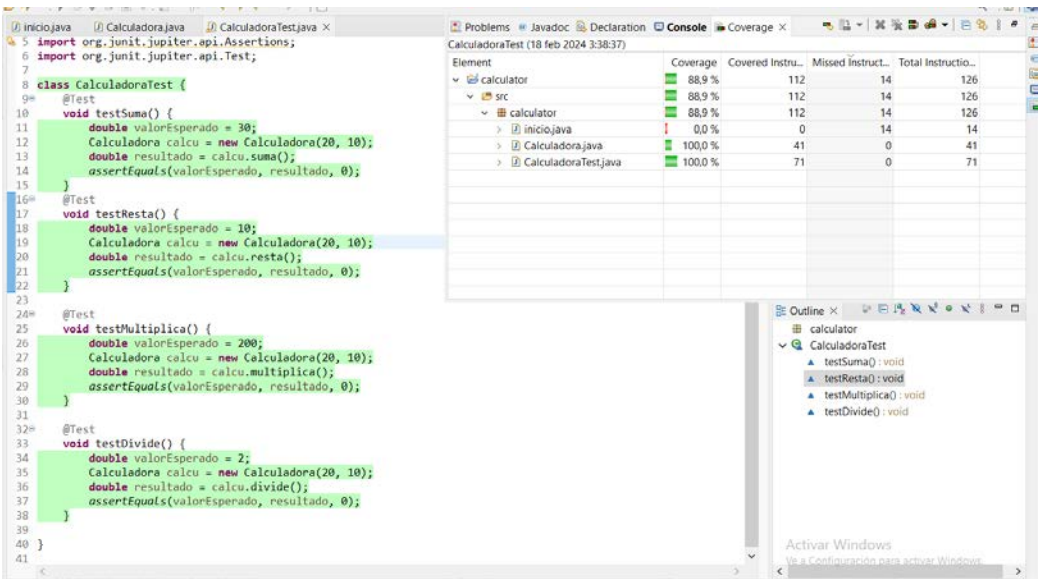
- @RunWith(JUnitPlatform.class): esta anotación é necesario incluíla. As clases e suites de proba anotadas con este etiqueta non se poden executar directamente como unha proba de "**JUnit 5**". Ditas clases e suites só pódense executar empregando a infraestrutura de "**JUnit 4**".
- @SelectPackages("nome do paquete"): se as clases a probar están dentro dun paquete, empregamos esta notación. O nome das clases ten que terminar coa palabra **Test** ou **Tests.**
- @SelectClasses({Class1Test.class, Class2test.class, ...): úsase esta anotación se se quere indicar o nome das clases que forman parte do conxunto de probas que son as que se van executar.
- @SuiteDisplayNome("Texto"): esta anotación emprégase para declarar un nome personalizado que executase como un conxunto. Este nome mostrarase ao executarse a suite.
- Dentro da clase que se crea non se xera ningunha liña de código.

6. Medición da cobertura do código

Unha vez executados os casos de proba, e verificado que todo funciona, a pregunta é "temos a seguridade de que temos cubertas todas as ramas de execución do programa?".

Eclipse permite visualizar a cobertura do código dos tests unitarios feitos empregando a ferramenta **EclEmma**. Isto determina que partes do software foron executadas polos casos de proba.

Accédese dende o menú Run>Coverage, que proporcionará a seguinte información:



- Coverage: porcentaxe cuberta na proba. En vermello móstrase o que non se probou, en verde o que si se ten probado.
- Covered Methods: métodos cubertos pola proba.
- Missed Methods: métodos non cubertos pola proba.
- Total Methods: número de métodos en total da clase.