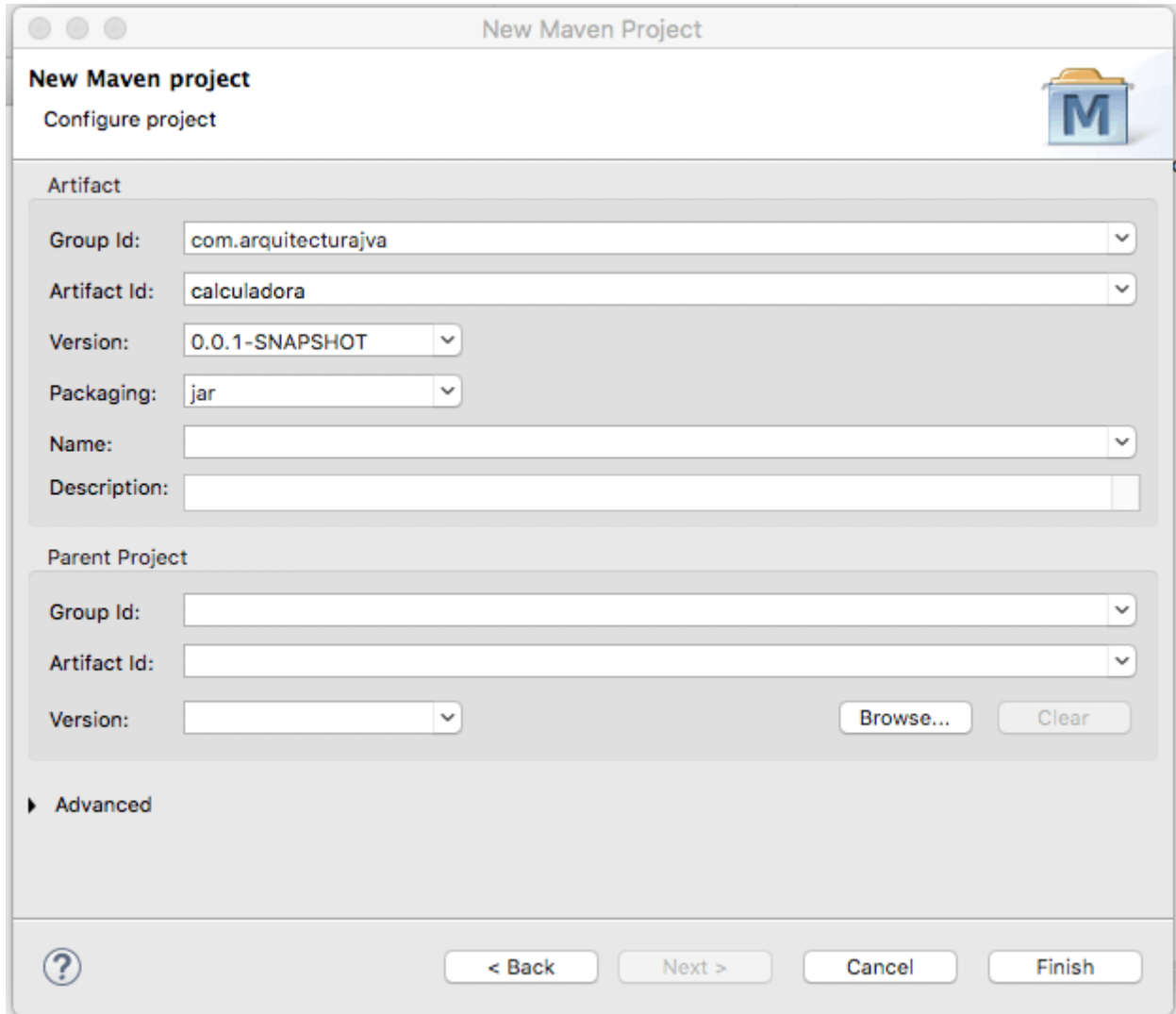


## LABORATORIO 2

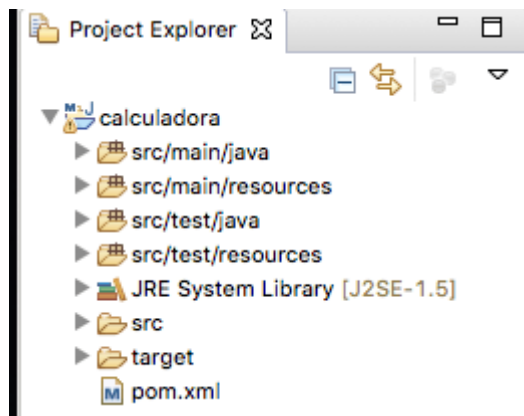
Vamos a construir con Eclipse un proyecto maven de una calculadora que sume 2 numeros.



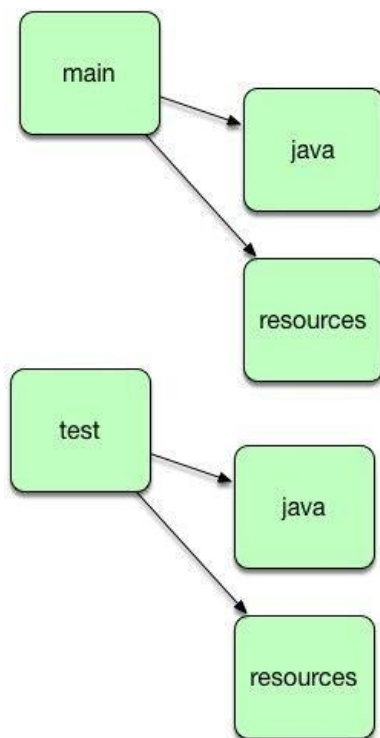
The screenshot shows the 'New Maven Project' dialog box in Eclipse. The title bar reads 'New Maven Project'. Below the title bar, there is a section titled 'New Maven project' with a sub-label 'Configure project' and a small icon of a folder with an 'M' on it. The dialog is divided into several sections:

- Artifact**: This section contains several input fields:
  - Group Id:** A text box containing 'com.arquitecturajava'.
  - Artifact Id:** A text box containing 'calculadora'.
  - Version:** A dropdown menu showing '0.0.1-SNAPSHOT'.
  - Packaging:** A dropdown menu showing 'jar'.
  - Name:** An empty text box.
  - Description:** An empty text box.
- Parent Project**: This section contains:
  - Group Id:** An empty text box.
  - Artifact Id:** An empty text box.
  - Version:** A dropdown menu.
  - Browse...** and **Clear** buttons.
- Advanced**: A section that is currently collapsed, indicated by a right-pointing triangle icon.

At the bottom of the dialog, there is a row of buttons: a help button (question mark icon), '< Back', 'Next >', 'Cancel', and 'Finish'.



Aquí mucha gente se queda un poco sorprendida , ya que el nombre de carpetas es un poco simplón. Mucha gente se espera una estructura “superior”. Pero resulta que solo existe main con java y resources y test con java y resources



Ya tenemos el proyecto construido pero no hemos construido código . Es momento de construir nuestra mini calculadora

```
public class Calculadora {
```

```
    public double sumar(double numero1,double numero2) {
```

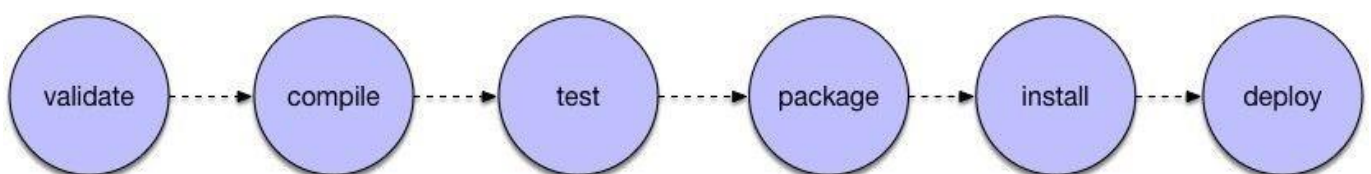
```
        return numero1+numero2;
```

```
    }
```

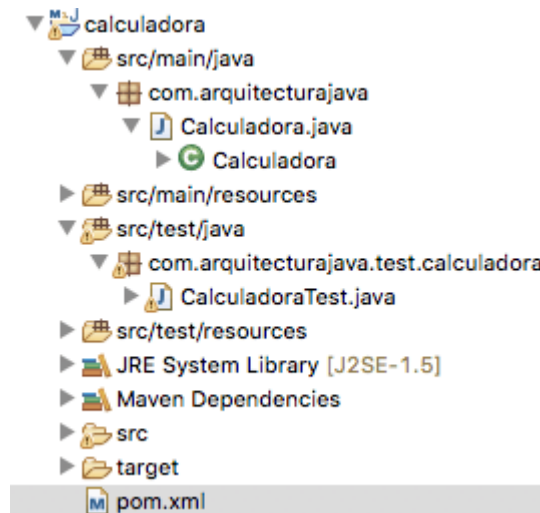
```
}
```

Ya tenemos nuestra super calculadora , sin embargo esto no es funcional ya que tendremos que compilarlo para poder ejecutarlo. Es aquí donde el concepto de Maven Artifact es capaz de abstraer de una forma correcta y a detalle lo que implica compilar un bloque de código. En principio un desarrollador piensa que compilar el código es simplemente pulsar al botón de compilar y se genera un compilado en nuestro caso un jar o algo similar.

La realidad es que esto no es tan directo el proceso de compilación y generación de un jar se puede dividir en bastantes fases



En esta caso vamos a añadir un sencillo test que nos permita comprobar que la calculadora suma de forma correcta los números.



El test es sencillo de construir:

```
package com.arquitecturajava.test.calculadora; import static  
org.junit.Assert.*;
```

```
import org.junit.Test;
```

```
import com.arquitecturajava.Calculadora;
```

```
public class CalculadoraTest { @Test
```

```
public void test() {
```

```
    assertEquals(4, Calculadora.sumar(2, 2),0);
```

```
}
```

```
}
```

En este caso al ser dos tipos dobles el método `assertEquals` recibe un parámetro adicional `delta` para asignar que precisión es la deseada. En este caso al pasarle un cero le

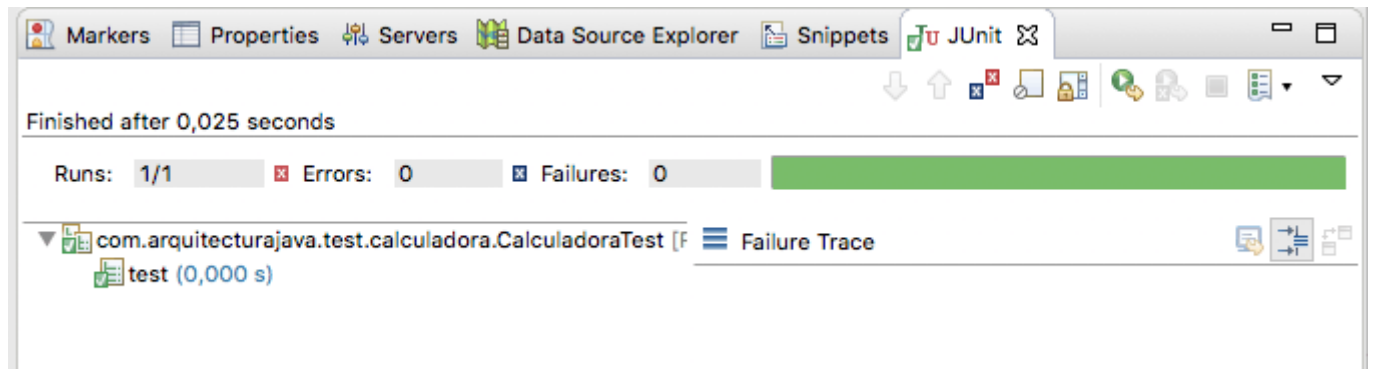
decimos que tiene que ser totalmente preciso. Estamos ante nuestro primer Maven artifact y este artefacto depende de otro artefacto , concretamente del de Junit ya que acabamos de construir una prueba unitaria. Así que deberemos modificar el fichero pom.xml para añadir dicha dependencia

```
<project      xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.arquitecturajava</groupId>
<artifactId>calculadora</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>

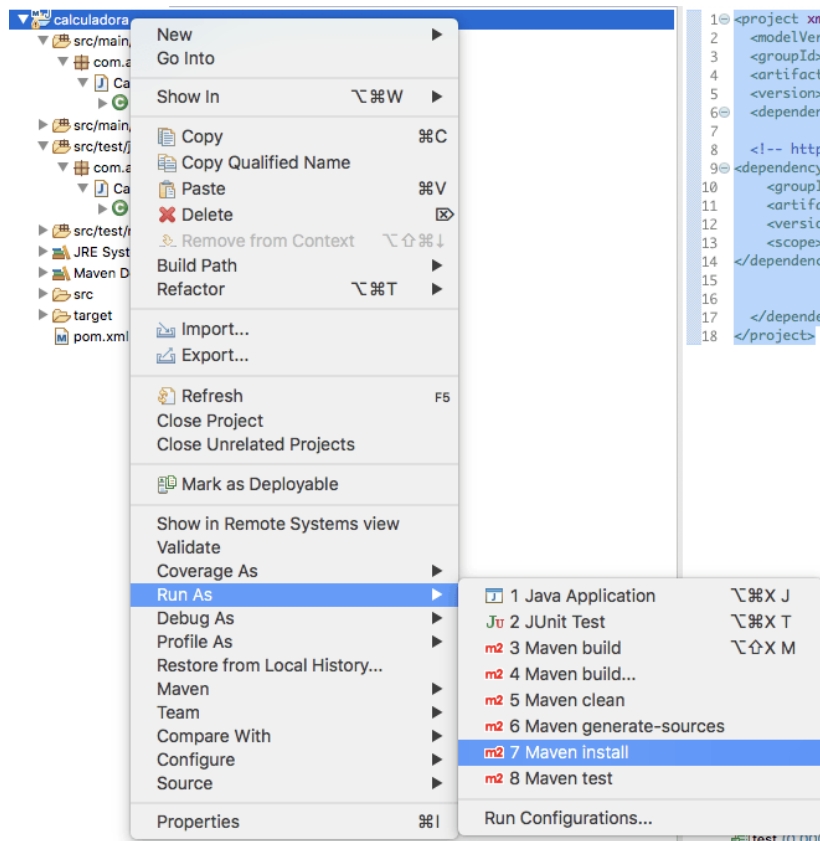
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>

</dependencies>
</project>
```

Una vez hecho esto ejecutamos el test



Una vez que los test se ejecutan correctamente podemos ejecutar en eclipse Maven Install e instalar nuestro artefacto en el repositorio local



Es momento de construir otro proyecto Maven que dependa del artefacto previamente construido y que hemos instalado en el repositorio. A este artefacto le llamaremos sumarcuadrado y se encarga de sumar dos números al cuadrado , para ello se apoyará en el artefacto previamente

compilado e instalado que es nuestra calculadora, veamos el fichero pom.xml.

```
<project      xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.arquitecturajava</groupId>
<artifactId>sumacuadrados</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>

<dependency>
<groupId>com.arquitecturajva</groupId>
<artifactId>calculadora</artifactId>
<version>0.0.1-SNAPSHOT</version>
</dependency>

<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

Como podemos ver hace uso también de las pruebas unitarias. El código de nuestra clase es muy sencillo:

```
package      com.arquitecturajava.cuadrados;      import
com.arquitecturajava.Calculadora;      public      class
SumaCuadrados {
public static double sumar(int numero1 , int numero2) {

return Calculadora.sumar(Math.pow(numero1, 2),
Math.pow(numero2, 2));
}

}
```

En este caso se apoya en la clase anterior para realizar la suma que se encuentra en otro artefacto y eleva los valores al cuadrado. Si vemos una prueba unitaria tendremos el siguiente código:

```
package com.arquitecturajava.cuadrados.test; import static
org.junit.Assert.*;
import org.junit.Test;
```



```
import

com.arquitecturajava.cuadrados.

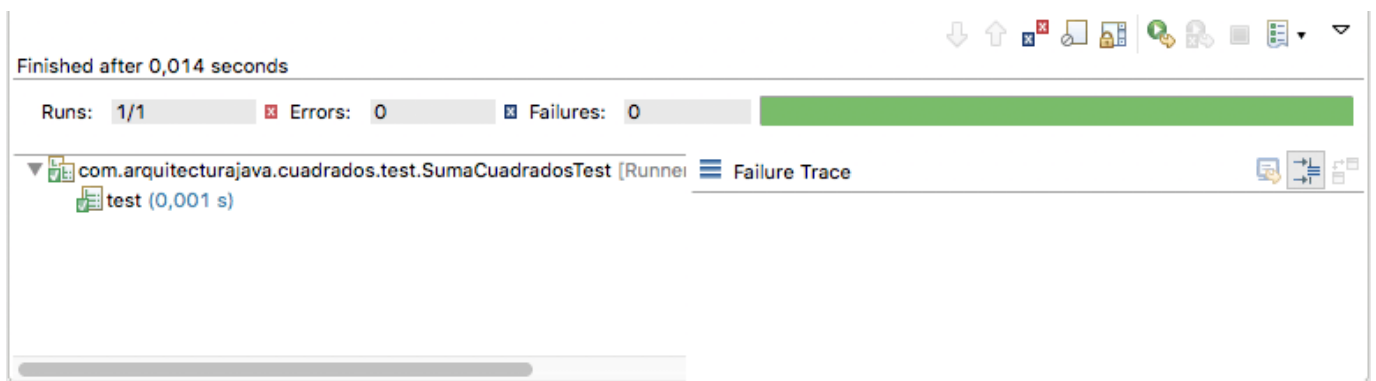
SumaCuadrados;    public    class

SumaCuadradosTest {

    @Test
    public void test() {
        assertEquals(8, SumaCuadrados.sumar
            (2, 2), 0);
    }

}
```

Ejecutamos el test :



Acabamos de hacer uso del artefacto de la calculadora en otro artefacto el de sumacuadrados. Hemos usado un bloque de código reutilizable. Así pues un artefacto no es ni más ni menos que una abstracción sobre el concepto de código reutilizable . Eso sí es una abstracción muy compleja y de entrada nada sencilla de entender , sin embargo esto es necesario para cubrir el gran número de situaciones diferentes que pueden aparecer.