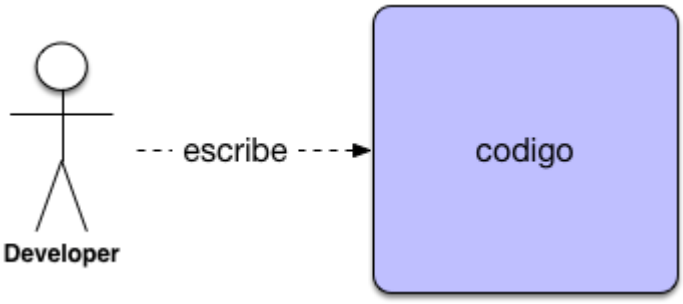
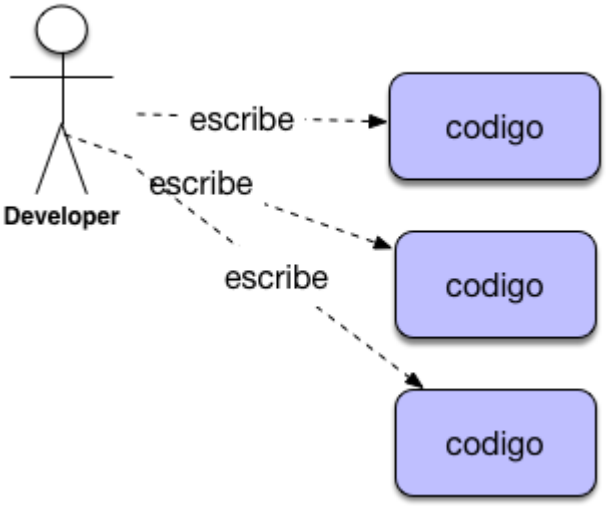
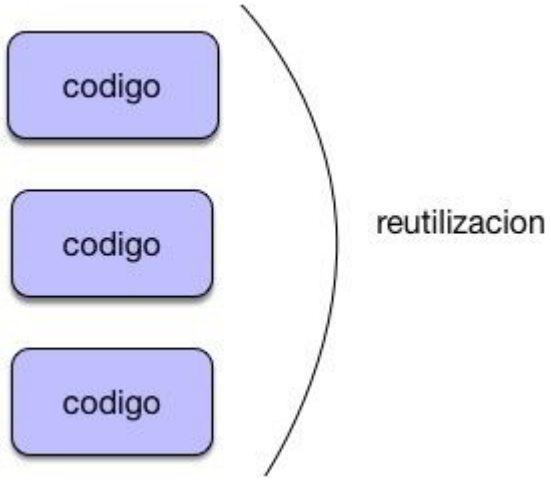
El concepto de Maven Artifact es uno de los conceptos que más cuesta entender cuando uno trabaja con Maven . ¿Qué es un Maven Artifact? . Explicarlo a veces no es sencillo . Pero si hablamos de programación a nivel general lo que estamos haciendo siempre es escribir código ,eso es lo que hacemos en el día a día.



¿Es suficiente para programar de forma correcta , simplemente escribir código ?. La realidad es que no . Para programar de una forma correcta por lo menos deberíamos construir código que sea reutilizable . Normalmente la reutilización esta fuertemente ligada con la modularidad. Es decir a bloques de código más pequeños mayor es la posibilidad de reutilización.



Un Maven Artifact no es ni más ni menos que un bloque de código reutilizable.



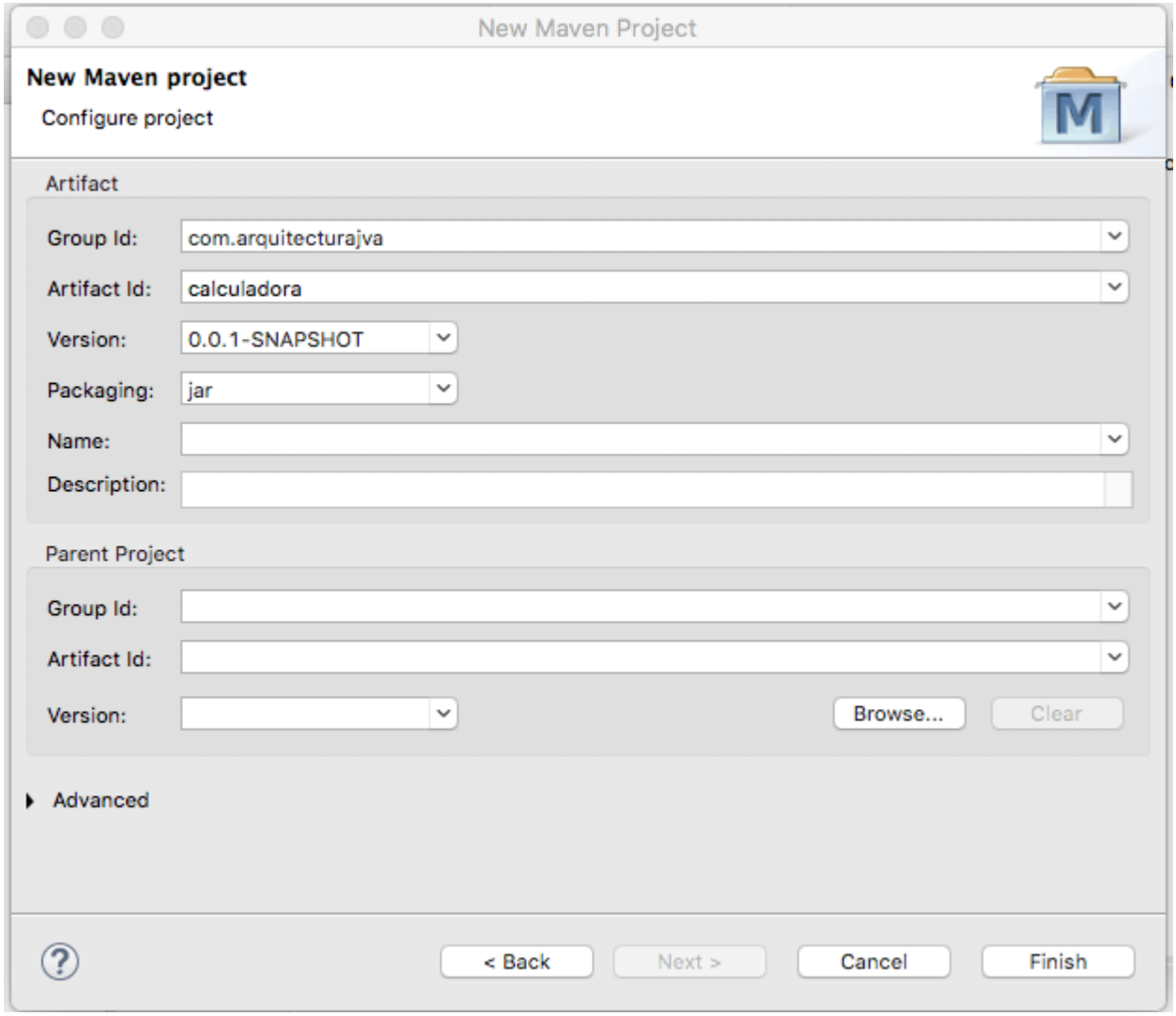
Muchas veces la gente me dice ahh entonces es lo mismo que un jar (java archive) o libreria. La realidad es que NO, no es lo mismo porque un jar es un bloque de código reutilizable pero ya compilado.

Ok ok entonces un Maven Artifact es un bloque de código fuente. No , la verdad es que tampoco es eso. ¿Entonces qué es? . Vamos a intentar explicarlo paso a paso. Para ello tenemos que entonces que entender en un primer momento que un Maven Artifact es una abstracción sobre el concepto de bloque de código reutilizable.

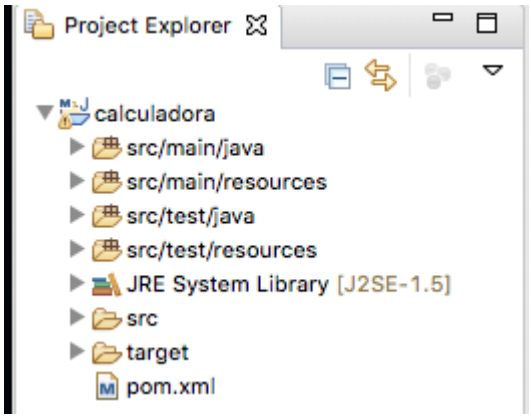
Si pensamos un poco en que es un bloque de código nos daremos que todo bloque de código cumple con algunos principios muy muy elementales.

1. Tiene un nombre o un identificador
2. Pertenece a una empresa o persona
3. Tiene una versión concreta

   Es cierto que esto es muy básico pero es lo esencial . Vamos a construir con Eclipse un proyecto maven de una calculadora que sume 2 numeros.

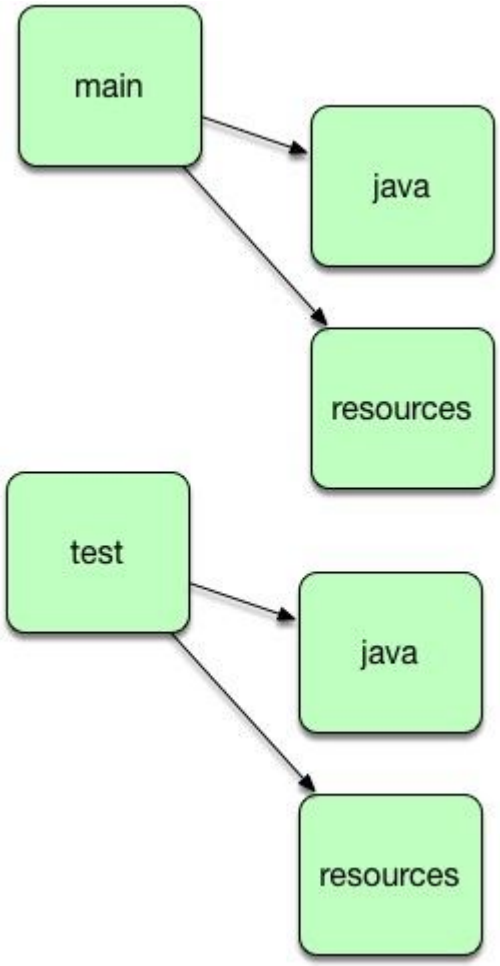


Ok es cierto que me pide esos datos cuando creamos el proyecto. Hasta aquí es entendible, pulsamos en finalizar y tendremos nuestro proyecto.



Maven Artifact Estructura

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



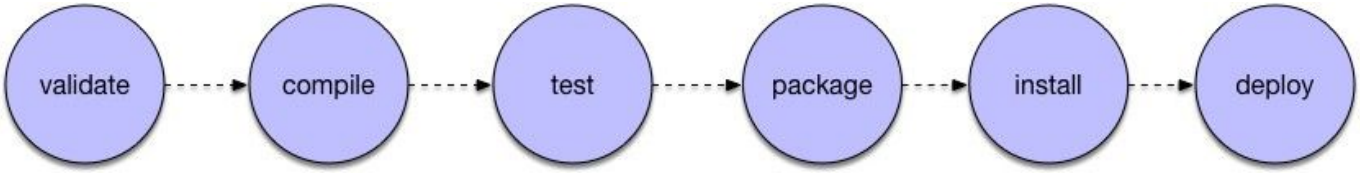
Ahora pensemos un poco fuera del mundo de Java ¿Qué comparten todos los coches a la hora de conducirlos? , únicamente el “volante” . Un coche de choques no tiene marchas por ejemplo y es un coche. Así que solo el VOLANTE. Esto es lo grande que tiene Maven la estructura es sencilla para soportar cualquier proyecto Java. Ok 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.

Maven Artifact life cycle

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



Es aquí donde mucha gente dice… ohhh ohhh.. abandonemos que esto se complica mucho. La realidad es un poco diferente se trata de un proceso muy simple pero hay que entenderlo, vamos a ello:

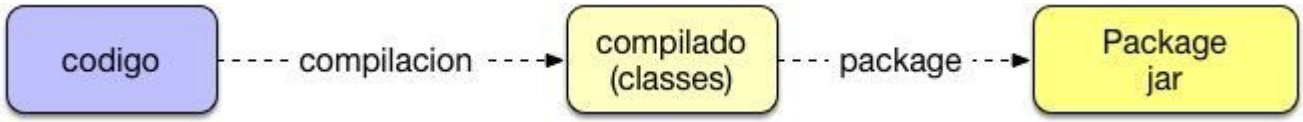
validate : Simplemente comprueba que el proyecto tiene la estructura correcta y los ficheros

están donde tienen que estar . Ok a nivel práctico hace poca cosa.

compile : Nos compila el código, este es fácil de entender , eso si nosotros compilamos el código primero pasará la por la fase de validate.

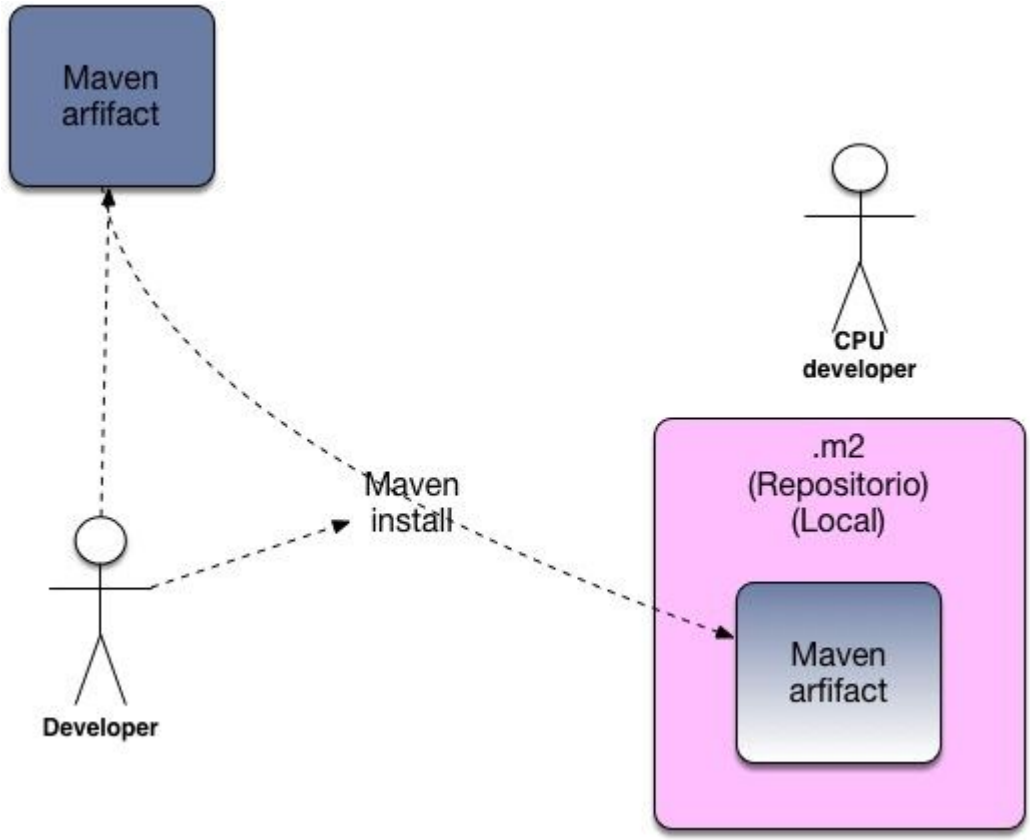
test: Se encarga de pasar las pruebas unitarias , algo que siempre debiéramos tener por lo tanto es otra fase importante y que Maven de alguna forma refuerza.

package: Recordemos que una cosa es compilar nuestro código y generar los ficheros .class y otra cosa muy diferente es generar un empaquetado que se pueda “reutilizar” . Recordemos el concepto de código reutilizable.

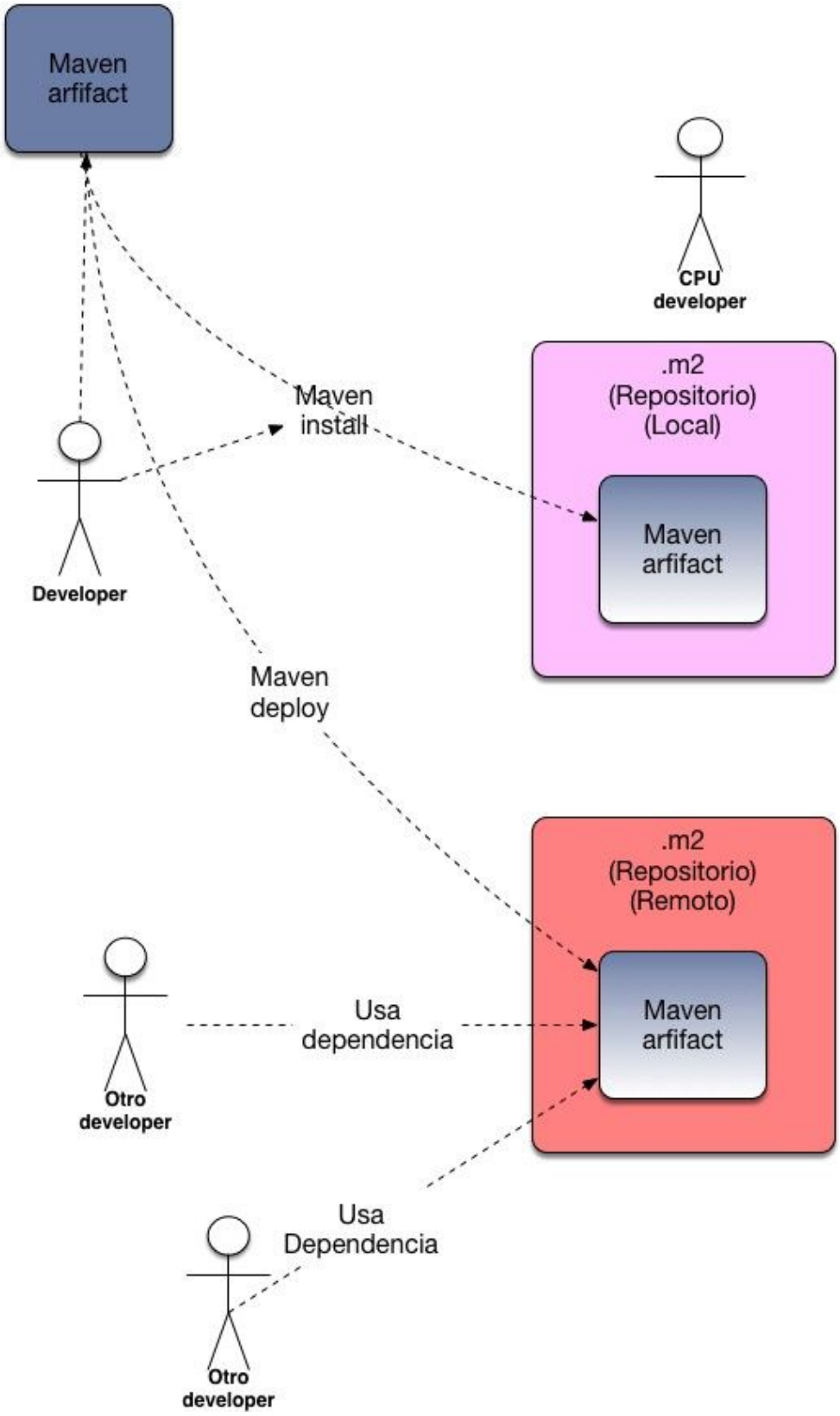


Así que realmente era bueno especificar las cosas un poco más. Compilar no es lo mismo que empaquetar . Aunque a veces las herramientas lo simplifican tanto que lo parece.

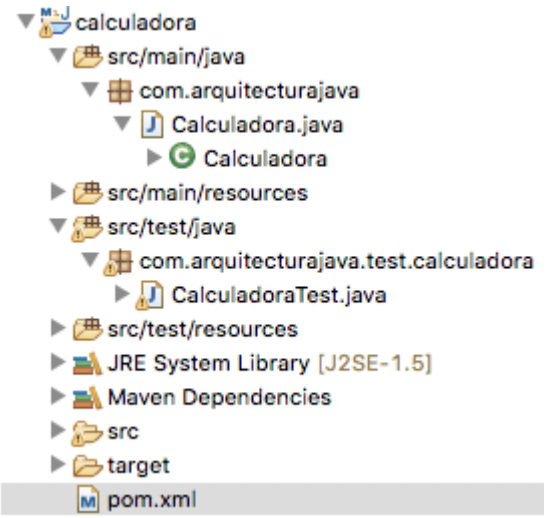
install: Aquí mucha gente se pierde ,¿install? oye yo ya tengo mi código empaquetado para que alguien lo use . ¿Se necesita algo más? … la realidad es que no…. ¿Estas seguro? … ¿es suficiente con pasar a alguien ese empaquetado?? . La realidad es que no la gente nos va a solicitar el empaquetado y el código fuente . El código fuente es necesario nos le van a solicitar siempre. Así que de alguna forma necesitamos instalar nuestro artefacto de maven en un repositorio de Maven. Eso es lo que se hace en la fase de install.



A partir de este momento tenemos instalado nuestro Maven artifact en un repositorio para su posterior utilización este repositorio se encuentra en la famosa carpeta .m2 . ¿Es esto suficiente ? . La realidad es que no ya que tendremos el artefacto instalado en nuestro repositorio local. Si queremos que nuestro artefacto pueda ser utilizado por otros developers necesitaremos realizar maven deploy esto nos lo instalará en un repositorio Maven remoto al que otros usuarios podrán acceder (Nexus o Artifactory).


Maven
arfifact
CPU
developer
.m2
(Repositorio)
(Local)
Maven
install
Maven
arfifact
Developer
Maven
deploy
.m2
(Repositorio)
(Remoto)
Maven
arfifact
Usa
dependencia
Otro
developer
Usa
Dependencia
Otro
developer

Una vez que nos ha quedado claro como funciona el ciclo de vida de un artefacto vamos a retomar el que nosotros habiamos construido , la calculadora. En esta caso vamos a añadir un sencillo test que nos permita comprobar que la calculadora suma de forma correcta los números.



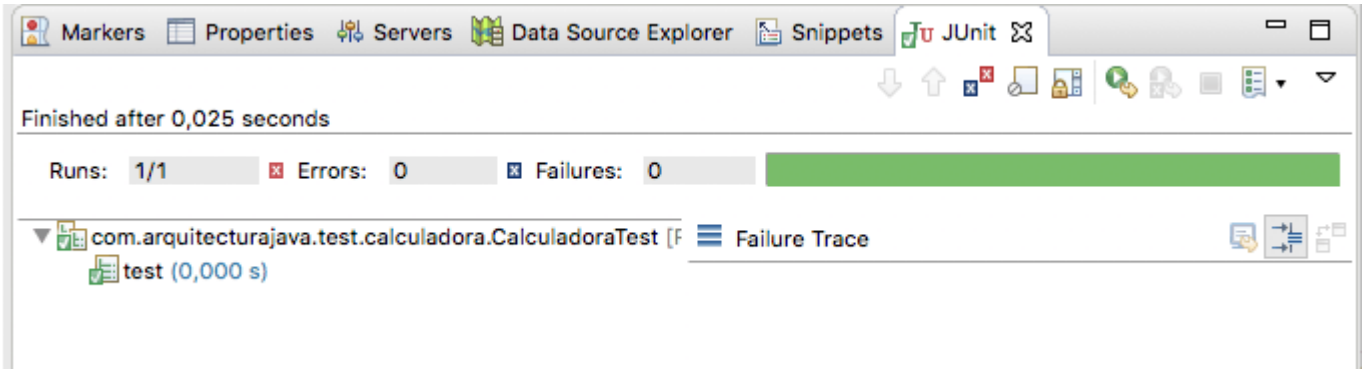
El test es sencillo de construir:

```
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);
        }

}
```

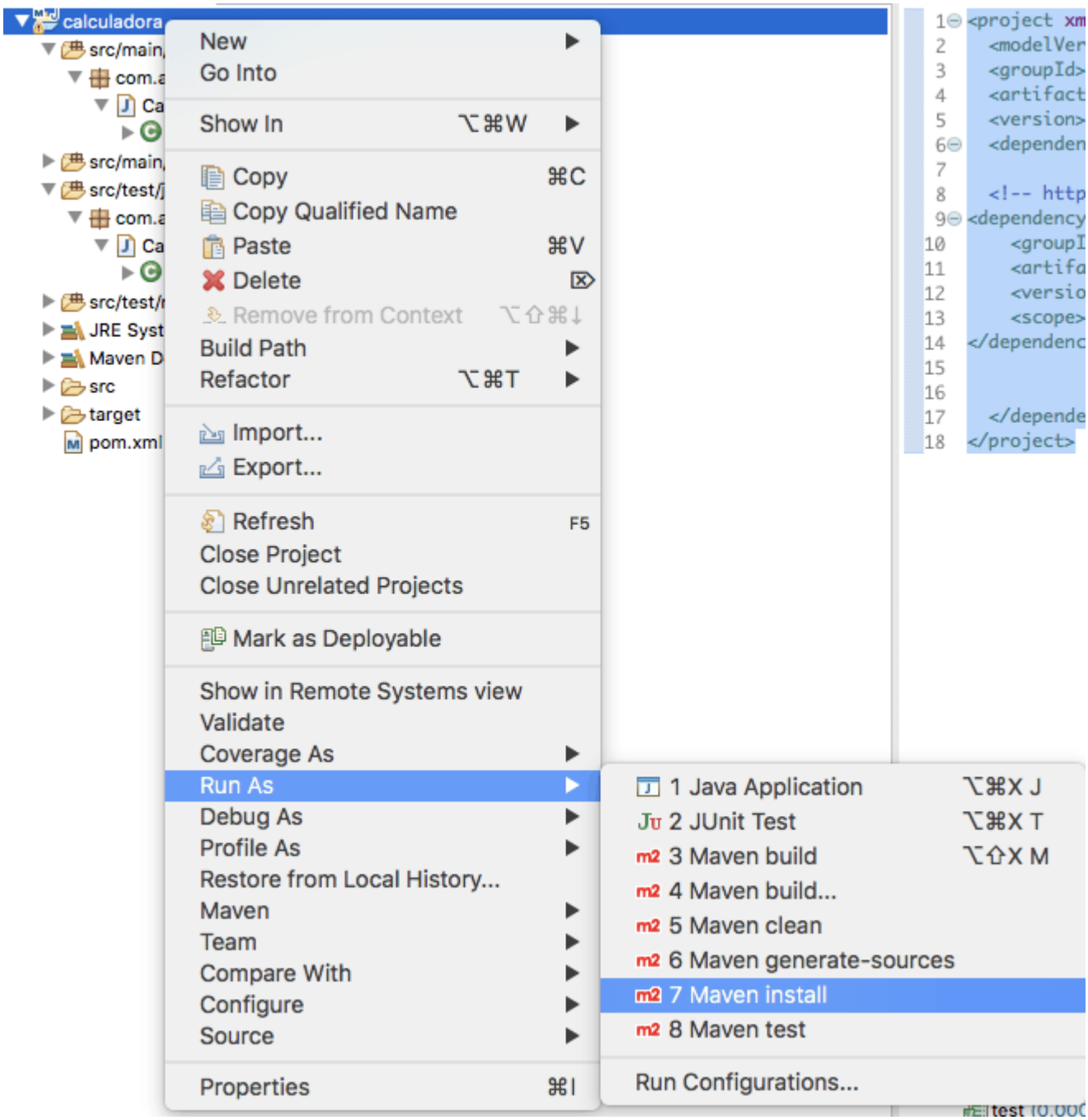
En este caso al ser dos tipos dobles el método assertEquals recibe un parámetro adicional delta para asignar que precisión es la deseada. En este caso al pasarle un cero le decimos que tiene que ser totalmente preciso. Estamos ante nuestro primer Maven artifact y este artefacto depende de otro artefacto , 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.arquitecturajva</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.



Usando un Maven Artifact

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:

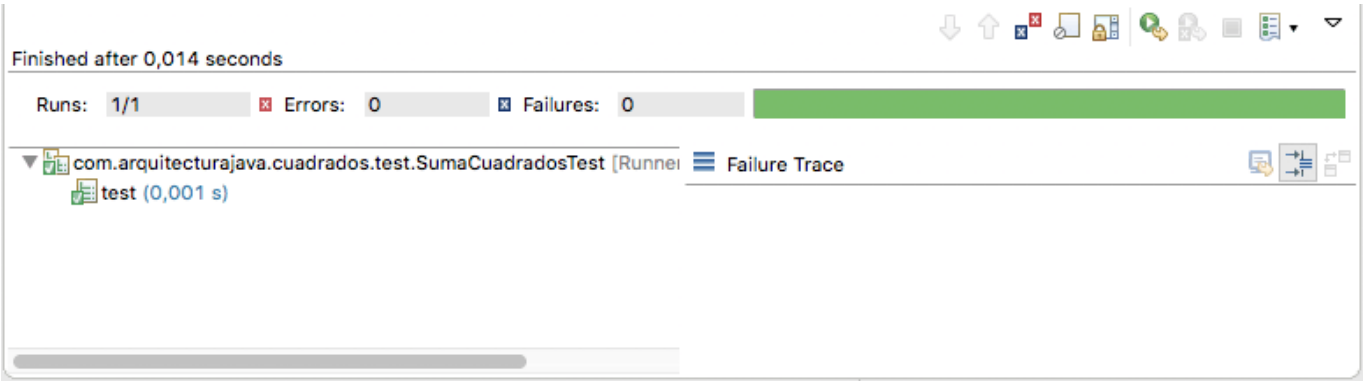
```
package com.arquitecturajava.cuadrados;

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.

Otros artículos relacionados

1. Maven Parent POM y uso de librerias

2. Utilizando Maven Profiles
3. Maven (I)
4. Maven (II)
5. Maven (III)

Artículos externos

1. Maven