

Blog sobre Java EE

Estás aquí: [Inicio/Arquitectura/Java 9 Modules y el concepto de modularidad](#)

Java 9 Modules y el concepto de modularidad

por [Cecilio Álvarez Caules](#) — [Deja un comentario](#)

	€10,21	€1,08	€0,70	€1,20	€10,09	€2,18	€3,09

Todavía nos quedará tiempo para usar **Java 9 Modules** ya que acaban de llegar. ¿Pero qué son y para que sirven los **Java 9 Modules**?. Hasta hoy en día Java ha organizado sus clases a través del concepto de paquetes que es un concepto puramente lógico. Un conjunto de **clases pertenecen a un paquete determinado**. Hasta aquí todo correcto . A nivel físico varios packages son ubicados en un **JAR** o Java Archive.



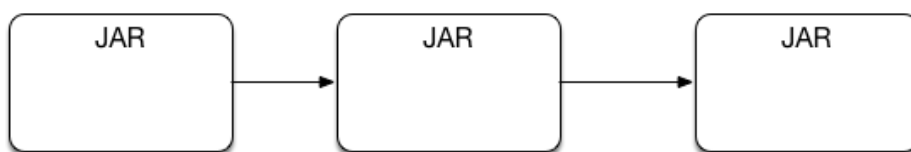
Esto ha terminado siendo un poco pobre ya que es necesario **tener más de organización y modularidad** a la hora de trabajar **con grupos de clases y sus dependencias**. Por ejemplo clases de un mismo paquete podrían estar ubicadas en dos JARs diferentes.



No solo eso sino que algunos de los ficheros JAR **a nivel de Java incluyen cientos de packages**. Por lo tanto estamos ante una situación que se acerca **bastante al concepto de monolito (una única pieza)**. Este es el caso mítico del **rt.jar** que agrupa a todas **las clases core de Java y sus packages**.

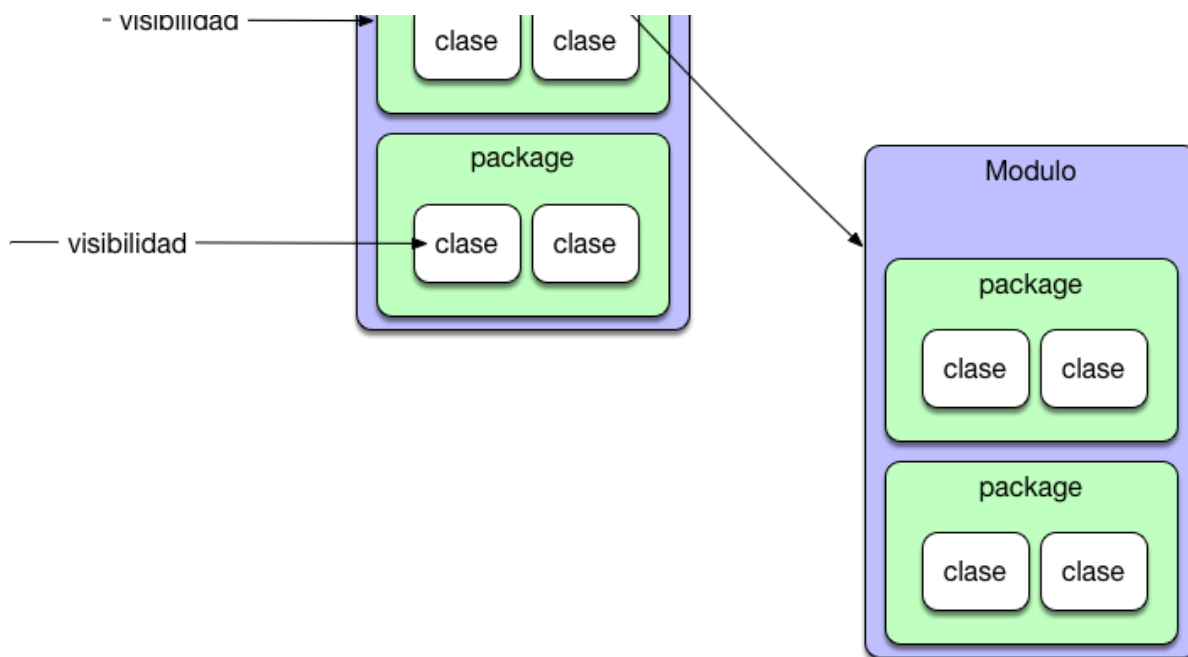


Otro de los problemas que siempre han existido es como gestionar las dependencias entre un JAR y otro con los packages que están asociados. Maven siempre ha ayudado a ello , pero es cierto que es un **herramienta aparte** , no algo propio del lenguaje.



Java 9 Modules

Para solventar **todos estos problemas Java 9 utiliza el concepto de módulo, algo que existe en otras plataformas como Node**. Un módulo es un conjunto de clases que pueden contener uno o varios packages y que define las dependencias con el resto de módulos **así como la visibilidad de las clases que contiene**.



Ejemplo de Java 9 Modules

Vamos a construir un proyecto en Eclipse en el cual veamos **un ejemplo sencillo de los módulos**. Para ello nos vamos a construir un **Utility Project**. Recordemos que un proyecto de utilidades define una librería o JAR. En este proyecto vamos a incluir tres ficheros (Factura, Utilidades y module-info).

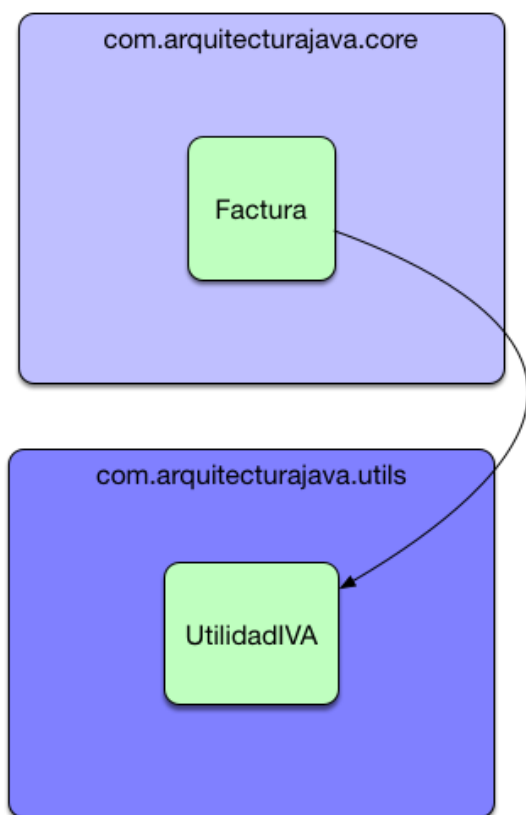
```

1  package com.arquitecturajava.core;
2  import com.arquitecturajava.utils.UtilidadIVA;
3
4  public class Factura {
5
6      private int numero;
7      private String concepto;
8      private double importe;
9      public int getNumero() {
10         return numero;
11     }
12     public void setNumero(int numero) {
13         this.numero = numero;
14     }
15     public String getConcepto() {
16         return concepto;
17     }
18     public void setConcepto(String concepto) {
19         this.concepto = concepto;
20     }
21     public double getImporte() {

```

```
1 package com.arquitecturajava.utils;  
2  
3 public class UtilidadIVA {  
4     public static double calcularIVA(double importe) {  
5         return importe *1.21;  
6     }  
7 }  
8  
9 }
```

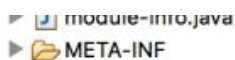
En este caso tenemos dos clases **Java relacionadas ubicadas en diferentes packages (core y utils)** .



Vamos a ver que información contiene el fichero que se encarga de la gestión de módulos.

```
1 module ModuloA {  
2     exports com.arquitecturajava.core;  
3 }
```

Es aquí donde podemos ver cual es la estructura de nuestro módulo.



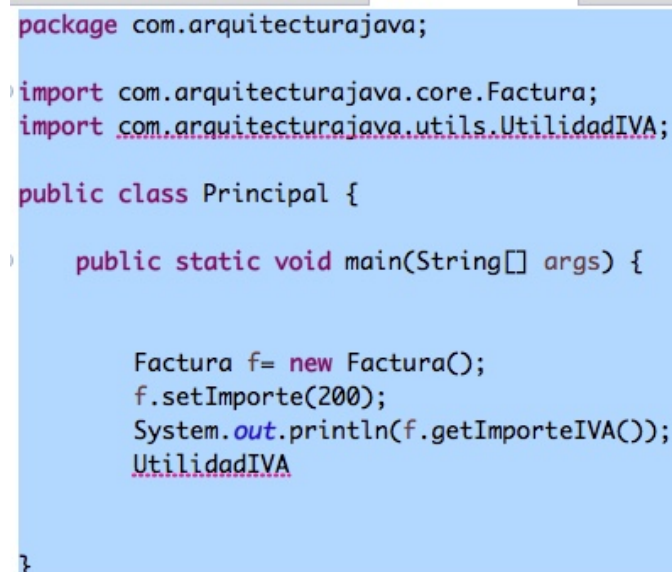
Es un módulo que no tiene dependencias **pero que como peculiaridad no exporta todos los packages**.

Únicamente se exporta el package **core que es el que contiene la clase Factura**. Es momento de usar nuestra librería en otro proyecto Java que tenga un fichero main.

```
1 package com.arquitecturajava;
2
3 import com.arquitecturajava.core.Factura;
4 import com.arquitecturajava.utils.UtilidadIVA;
5
6 public class Principal {
7
8     public static void main(String[] args) {
9
10
11         Factura f= new Factura();
12         f.setImporte(200);
13         System.out.println(f.getImporteIVA());
14         UtilidadIVA
15
16
17     }
18 }
```

Java 9 y acceso

En principio el código parece correcto pero si lo miramos en Eclipse nos mostrará lo siguiente:



```
package com.arquitecturajava;

import com.arquitecturajava.core.Factura;
import com.arquitecturajava.utils.UtilidadIVA;

public class Principal {

    public static void main(String[] args) {

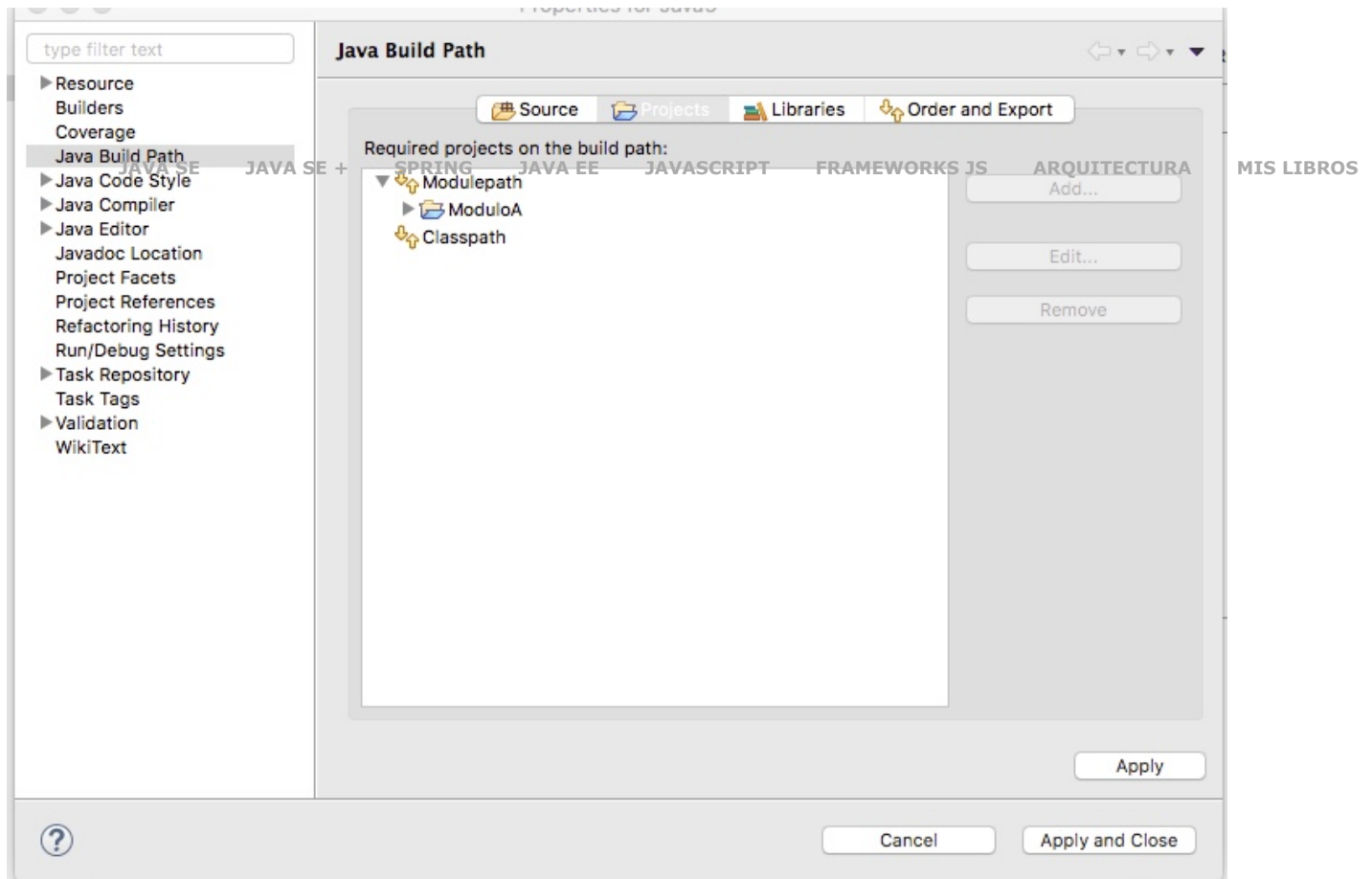
        Factura f= new Factura();
        f.setImporte(200);
        System.out.println(f.getImporteIVA());
        UtilidadIVA

    }
}
```

Este sitio web utiliza cookies para que usted tenga la mejor experiencia de usuario. Si continúa navegando está dando su consentimiento para la aceptación de las mencionadas cookies y la aceptación de nuestra [política de cookies](#), pinche el enlace para mayor información.

ACEPTAR

[plugin cookies](#)



Aún así no podemos acceder a UtilidadIVA ya que lo hemos cerrado explícitamente. Acabamos de construir nuestro primer ejemplo de **Java 9 Modules**. Esta tecnología es nueva pero afectará de forma importante a todas las aplicaciones en el futuro.

1. [Java 9 Collections y sus novedades](#)
2. [Java Stream Sum y Business Objects](#)
3. [Java 8 interface static methods y reutilizacion](#)
4. [Java Modules](#)

24

1

25 COMPARTIR

Televisión de calidad por 10€ al mes. Sin permanencia

Pruébalo gratis


sky

Aviso legal

Este sitio web utiliza cookies para que usted tenga la mejor experiencia de usuario. Si continúa navegando está dando su consentimiento para la aceptación de las mencionadas cookies y la aceptación de nuestra [política de cookies](#), pinche el enlace para mayor información.

[plugin cookies](#)
[ACEPTAR](#)

Notify of	<input type="text" value="new follow-up comments"/>	<input type="text" value="Email"/>	<input type="button" value="➤"/>
-----------	---	------------------------------------	----------------------------------



Start the discussion

BUSCAR

CUPÓN DESCUENTO NAVIDAD

Aprovecha mi cupón de Navidades para obtener un 50% de descuento **en** mis cursos de Java 😊

Cupón:NAVIDAD_2017

Mis Cursos de Java Gratuitos

Java Herencia

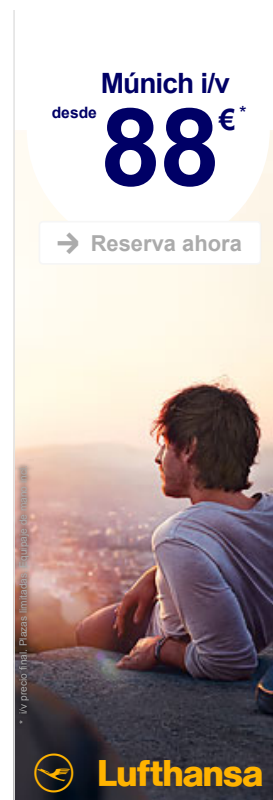


Java JDBC



Servlets





POPULAR

[Maven Parent POM y uso de librerías](#)

[Java Interfaces y el concepto de simplicidad](#)

[Java Generic Repository y JPA](#)

[Spring 5 Hello World](#)

[Spring Boot WAR sin Microservicios](#)

[Spring GetMapping ,PostMapping etc](#)

[PostMan App con Spring REST](#)

[Java Stream map y estadísticas](#)

[Spring REST Client con RestTemplates](#)

[Spring Boot Properties utilizando @Value](#)

CONTACTO

contacto@arquitecturajava.com

Este sitio web utiliza cookies para que usted tenga la mejor experiencia de usuario. Si continúa navegando está dando su consentimiento para la aceptación de las mencionadas cookies y la aceptación de nuestra [política de cookies](#), pinche el enlace para mayor información.

[plugin cookies](#)

[ACEPTAR](#)

[Java Collections Map y Set](#)[Java Iterator vs ForEach](#)[Java 9 Collections y sus novedades](#)[Introducción a Servicios REST](#)[¿Cuales son las certificaciones Java?](#)[¿Qué es Gradle?](#)[Ejemplo de JPA , Introducción \(I\)](#)[Ejemplo de Java Singleton \(Patrones y ClassLoaders\)](#)[Usando el patron factory](#)[REST JSON y Java](#)[Uso de Java Generics \(I\)](#)[Java Override y encapsulación](#)[Mis Libros](#)[¿Qué es un Microservicio?](#)[Comparando java == vs equals](#)[Spring MVC Configuración \(I\)](#)[Spring REST Client con RestTemplates](#)

Copyright © 2018 · [eleven40 Pro Theme](#) en [Genesis Framework](#) · [WordPress](#) · [Acceder](#)