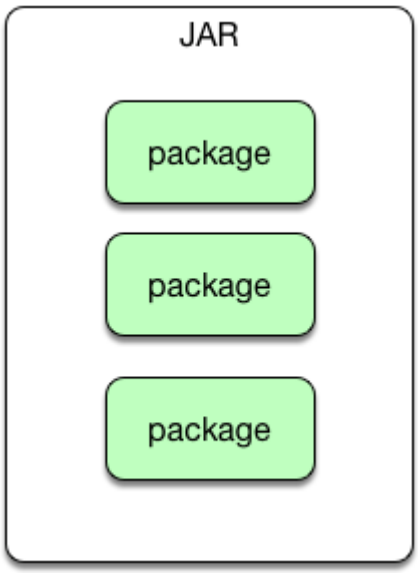
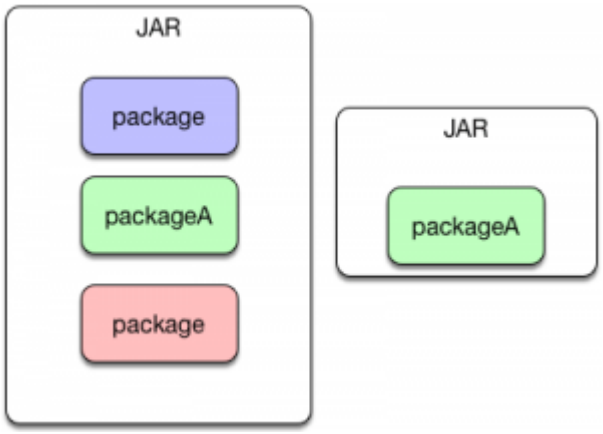
Tabla de Contenidos

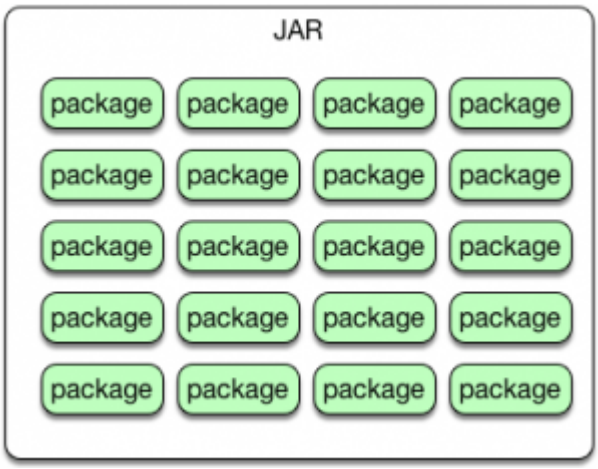
Todavía nos quedará tiempo para usar Java 9 Modules ya que acaban de llegar. ¿Pero qué son y para que sirven los Java 9 Modules?. Hasta hoy en día Java ha organizado sus clases a través del concepto de paquetes que es un concepto puramente lógico. Un conjunto de clases pertenecen a un paquete determinado.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



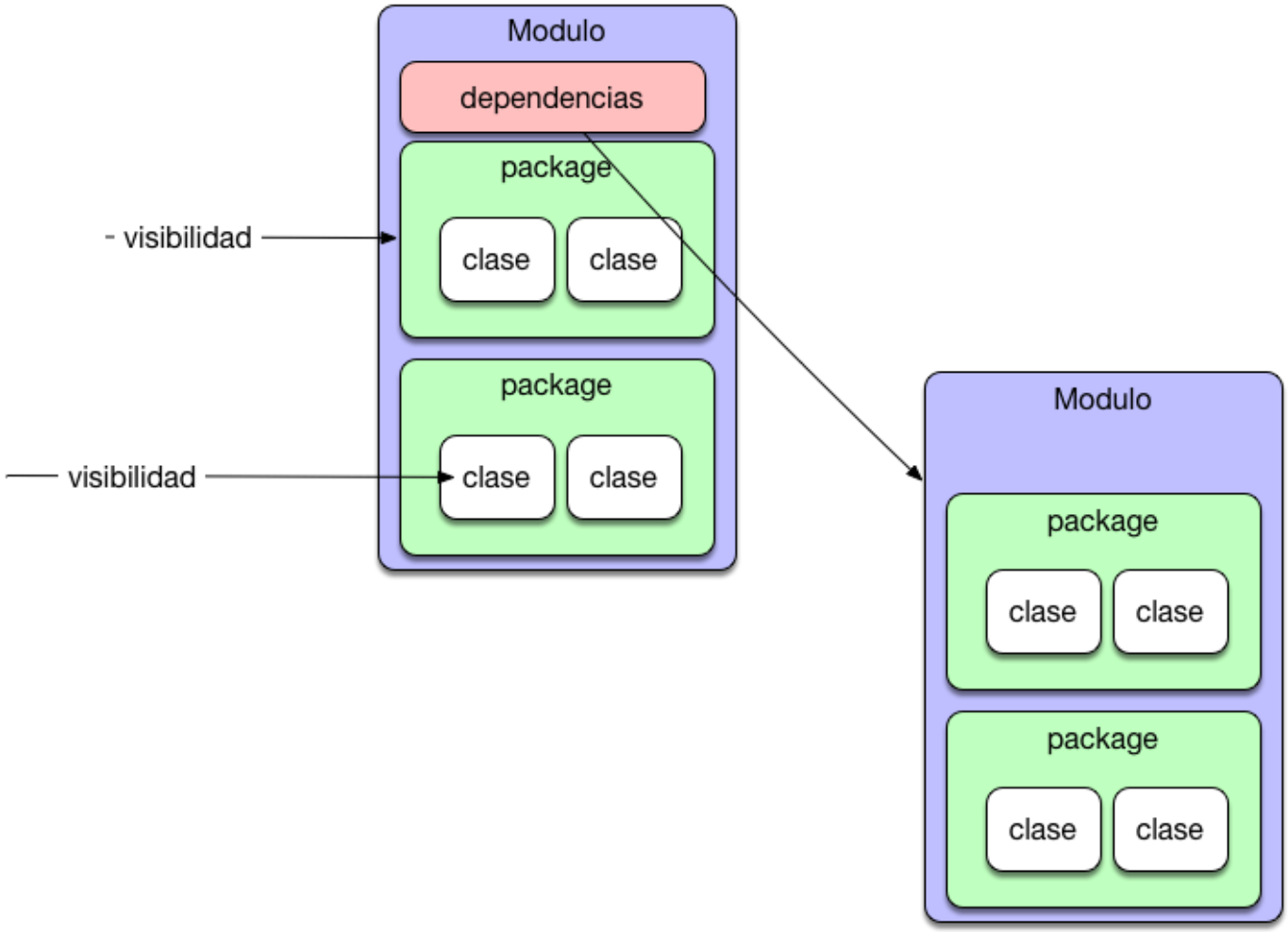
No solo eso sino que algunos de los ficheros JAR a nivel de Java incluyen cientos de packages. Por lo tanto estamos ante una situación que se acerca bastante al concepto de monolito (una única pieza). Este es el caso mítico del rt.jar que agrupa a todas las clases core de Java y sus packages.



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

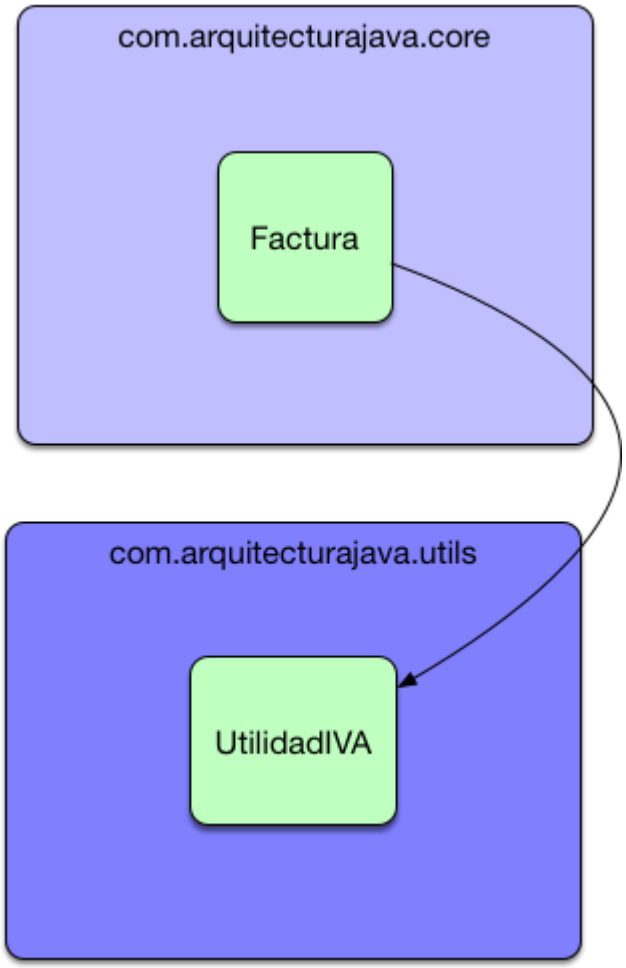
Vamos a construir un proyecto en Eclipse en el cual veamos un ejemplo sencillo de los módulos. Para ello nos vamos a construir un Utility Project .Recordemos que un proyecto de utilidades define una librería o JAR. En este proyecto vamos a incluir tres ficheros (Factura, Utilidades y module-info).

```
package com.arquitecturajava.core;
import com.arquitecturajava.utils.UtilidadIVA;

public class Factura {

	private int numero;
	private String concepto;
	private double importe;
	public int getNumero() {
		return numero;
	}
	public void setNumero(int numero) {
		this.numero = numero;
	}
	public String getConcepto() {
		return concepto;
	}
	public void setConcepto(String concepto) {
		this.concepto = concepto;
	}
```

```
        public double getImporte() {
                return importe;
        }
        public void setImporte(double importe) {
                this.importe = importe;
        }
        public double getImporteIVA() {
                return UtilidadIVA.calcularIVA(this.importe);
        }
}
```

```
package com.arquitecturajava.utils;

public class UtilidadIVA {

        public static double calcularIVA(double importe) {
                return importe *1.21;
        }
}
```

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.

```
module ModuloA {
 exports com.arquitecturajava.core;
}
```

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.

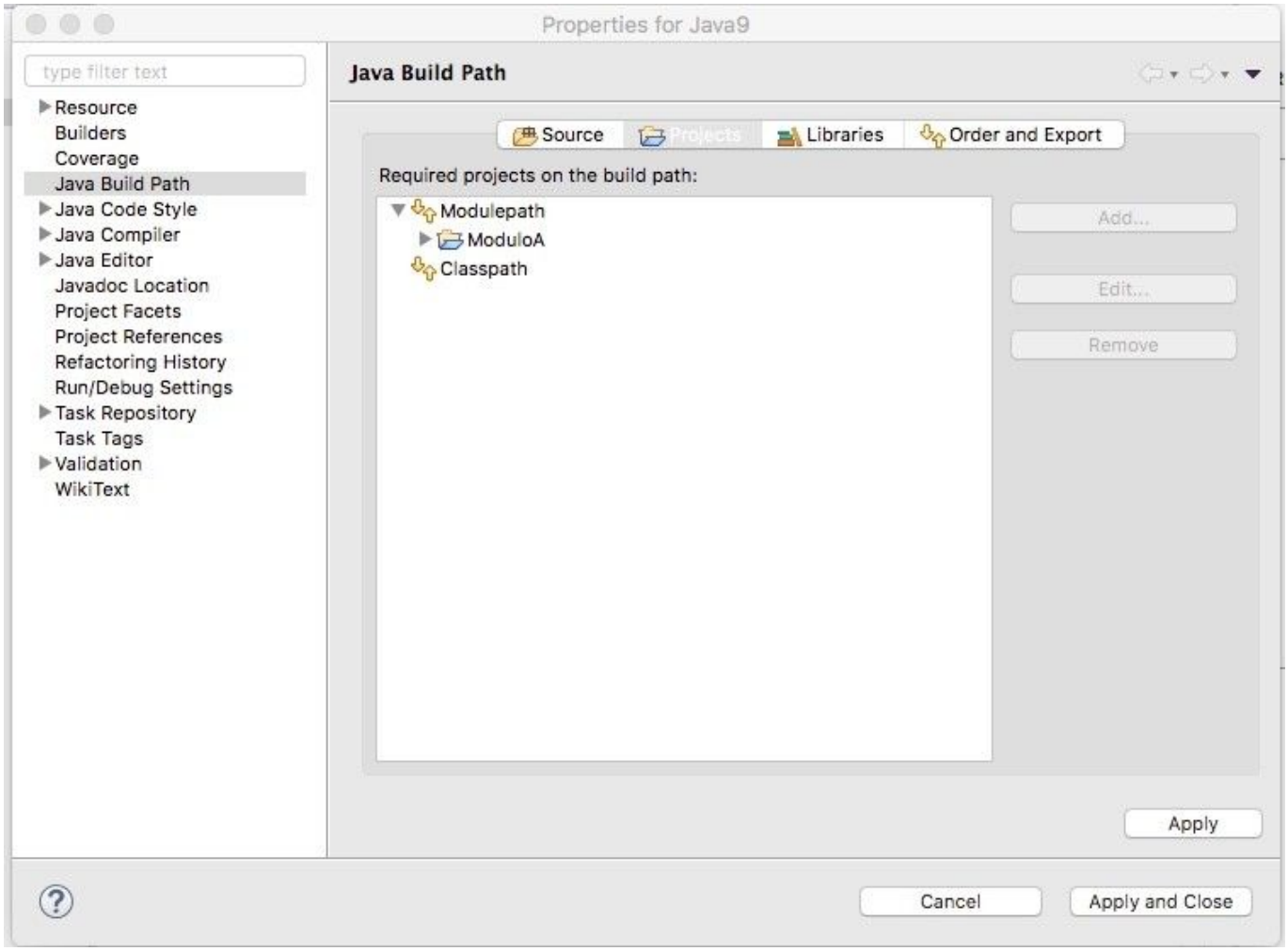
```
package com.arquitecturajava;

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

}
}
```

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


}
```

El código no funciona porque no podemos hacer uso de la clase UtilidadIVA ya que se encuentra en un package que el módulo no publica. Para poder utilizar el otro package hemos tenido que añadir a nuestro proyecto main la dependencia del módulo.



Aún así no podemos acceder a UtilidadIVA ya que lo hemos cerrado explicitamente. Acabamos de construir nuestro primer ejemplo de Java 9 Modules.

Java 9 Modules Export

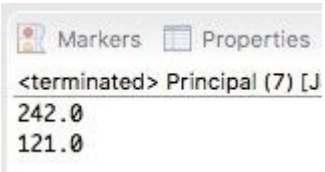
Es ahora cuando podemos replantearnos si la encapsulación que hemos hecho de nuestros packages es la más correcta. En este caso es evidente que no. El calculo del IVA es una operación que puede ser usada por otras librerías o por el programa principal. Por lo tanto podemos modificar nuestro módulo y publicarla.

```
module ModuloA {
        exports com.arquitecturajava.core;
        exports com.arquitecturajava.utils;
}
```

De esta forma ya podremos utilizar el código del calculo del IVA en un programa principal.

```
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());
                System.out.println(UtilidadIVA.calcularIVA(100));

}
}
```

El programa ya compila correctamente y si le ejecutamos nos devuelve los IVAs.



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