

	Ámbito de acceso			
	clase	+paquete	+subclases	+todo
private	X			
package	X	X		
protected	X	X	X	
public	X	X	X	X

```
private int enteroPrivado=7;
character characterPackage='X';
protected void metodoProtegido() {...}
public double metodoPublico() {...}
```

Los ámbitos de acceso son aplicables a las clases, y a sus componentes (campos y métodos), si bien en el caso de las clases, evidentemente sólo tienen sentido los ámbitos “public” y “package”.

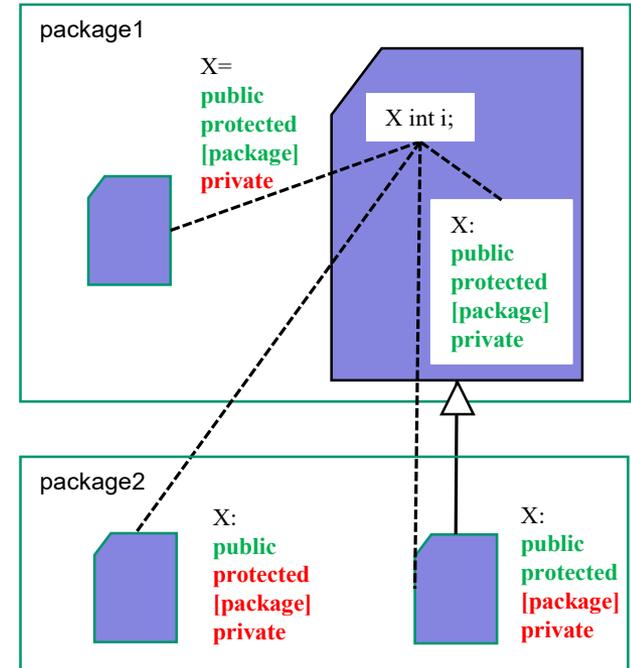
Palabras reservadas en Java				
abstract	assert ^(1,4)	boolean	break	byte
case	catch	char	class	const*
continue	default	do	double	else
enum ^(5,0)	extends	final	finally	float
for	goto*	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp ^(1,2)	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

```
1- //
2- // Aplicación ejemplo "HolaMundo"
3- //
4-
5- public class HolaMundo {
6-     public static void main(String[] args) {
7-         System.out.println("Hola, mundo");
8-     }
9- }
```

```
1- package packA;
2-
3- public class A {
4-     protected int i;
5-     // resto de definición de la clase
6- }
```

El acceso “privado” debe “abrirse” a “package” si queremos que las extensiones de la clase no vean imposibilitada la actuación sobre el elemento

```
1- package packB;
2-
3- import packA.A;
4-
5- public class B extends A {
6-     void unMetodo(A a, B b) {
7-         a.i = 10; // Ilegal. a pertenece a una clase de otro paquete.
8-         b.i = 10; // Legal. b puede acceder a su miembro i heredado de la definición de A
9-     }
10- // resto de la definición de la clase
11- }
```



Posibilidades de acceso a una variable en función de su atributo de acceso (X)

Comentario: la importancia de “getters” y “setters”

Java 9 refuerza el encapsulamiento: los módulos

Módulo= conjunto de paquetes diseñado para ser reutilizado

Nos limitaremos a una visión muy superficial.

En [Java Magazine sept/oct de 2017](#), a partir de la página 18, se da una explicación detallada de los módulos así como de sus motivaciones y el modo de ser propuestos e incluidos en una versión

```
El fichero module-info.java
module abc.xyz{
    exports com.foo.bar;
}
```

servicios

Palabras clave restringidas (fuera de la declaración de módulos no son palabras clave)

exports, module, open, opens, provides, requires, uses, with, to, transitive

requires [p] - (dependencia) - se utiliza material de otro módulo

Cuando “A requires B” se dice que “A lee B” y “B es leído por A”

requires transitive [p] - id. “requires” y transmite la dependencia a quien lea el módulo

Cuando “A requires transitive B”, si “Z requires A” implícitamente “Z requires B”

exports [p] – pone a disposición su material público y protegido.

exports [p] to [q,r,...,t] – (qualified export) - id. “export” limitando lectores.

uses [X] – (consume servicio) – utiliza objetos de una clase que concreta o implementa X

provides [X] with [C] – (proporciona servicio) – pone a disposición la clase C (servicio) que implementa o extiende X

El modulo será un “proveedor de servicio”.

opens [p] – permite acceso en tiempo de ejecución. (clases públicas y protegidas accesibles)

En consecuencia permite la introspección, que queda imposibilitada si no hay “export” o “opens”

opens [p] to [q,r,...,t] – id “opens” limitando módulos con acceso.

open – atributo de módulo indicando que todos sus paquetes son abiertos

open module modulename { ... }

A, B – módulos
p,q,r,t – paquetes
X – clase abstracta o interfaz
C – clase concreta