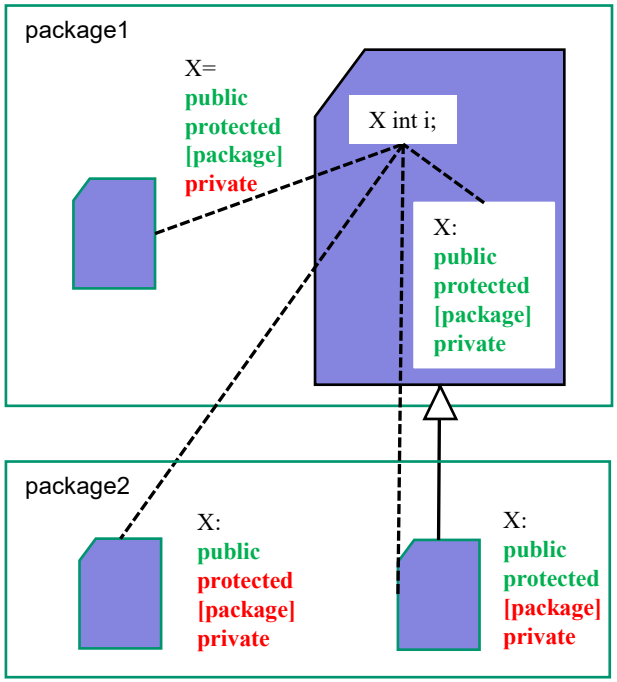
	Ámbito de acceso			
	clase	+paquete	+subclases	+todo
private	X			
package	X	X		
protected	X	X	X	
public	X	X	X	X

```
private   int    enteroPrivado=7;
          char   caracterPackage='X';
protected void   metodoProtegido(){…}
public    double metodoPublico() {…}
```

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

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`".

```
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-  }
```



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

```
package packA;

public class A {
  protected int i;
  // resto de definición de la clase
}
```

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

```
package packB;

public class B extends packA.A {
  void unMetodo() {
    this.i=10;
   ((packA.A)this).i=10;
  }
  static void otroMetodo(packA.A a, packB.B b) {
   a.i = 10;
    b.i = 10;
  }
  // resto de definición de la clase
}

class C {
  void unMetodo(packA.A a, packB.B b) {
   a.i = 10;
   b.i = 10; }
  // resto de definición de la clase
}
```

¡OJO!
La reescritura de un método no puede ser menos accesible que el método heredado.

```
public class A {
  public void m(){}
}

class B extends A{
 private void m(){}
}
```

Refinamiento de lo visto anteriormente:
En realidad en un mismo fichero podemos definir más de una clase, pero sólo una podrá ser pública, y será esta la que determine el nombre del fichero → lo veremos en detalle más adelante.

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" implicitamente "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

SÓLO A MODO DE COMENTARIO