Índice de contenido

1. Paquetes en Java

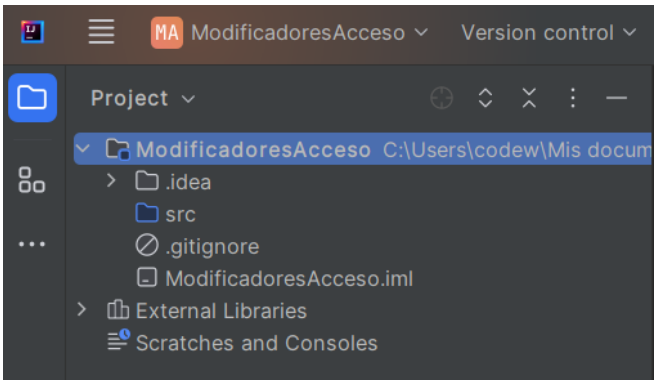
En Java, un paquete es un mecanismo utilizado para organizar y agrupar clases relacionadas. Los paquetes proporcionan un espacio de nombres para las clases, lo que ayuda a evitar conflictos de nombres y facilita la organización del código.

En términos simples un paquete es igual a una carpeta en el explorador de archivos. Asi como utilizamos carpetas para tener nuestra información mas organizada en nuestro computador, de igual manera empleamos paquetes para organizar las clases en nuestro programa.

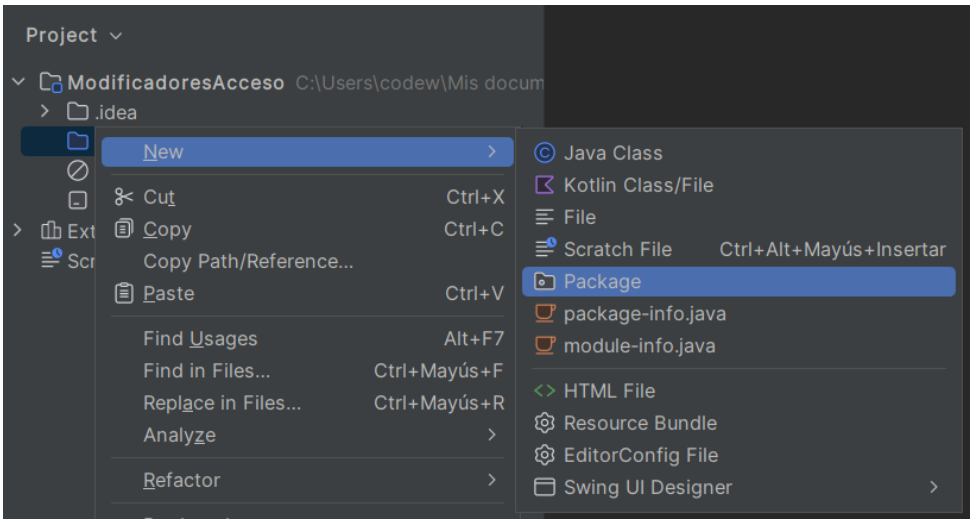
En programas mas complejos y empresariales pueden existir cientos de clases en un mismo proyecto por lo que seria demasiado complicado tener todas las clases en “src” de nuestro proyecto.

1.1. Ejemplo: Creación de un paquete desde IntelliJ

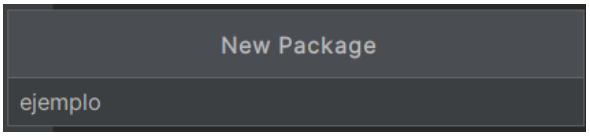
Creamos un nuevo proyecto en IntelliJ llamado “ModificadoresAcceso”.



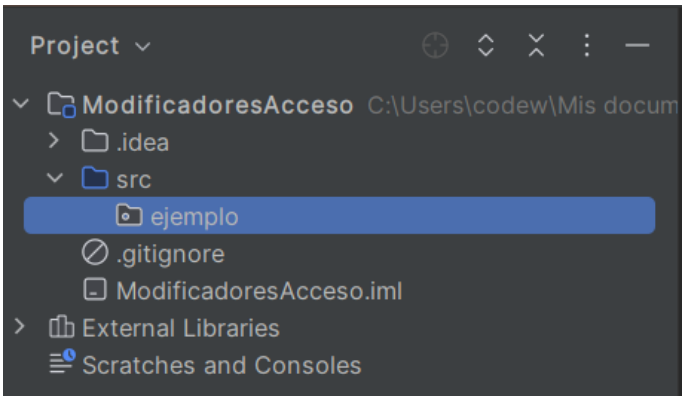
Damos click derecho sobre la carpeta “src” y seleccionamos New -> Package



Le asignamos el nombre “ejemplo” a nuestro paquete



De esta manera ya tendremos un nuevo paquete dentro de nuestro paquete “src”



2. Modificadores de acceso

2.1. ¿Qué son los modificadores de acceso?

Los modificadores de acceso en Java son palabras clave que se utilizan para controlar el acceso a las clases, atributos, métodos y constructores en una aplicación Java (la estructura básica de una clase y estos conceptos serán explicados a detalle en la clase 7). Estos modificadores determinan desde dónde se puede acceder a un miembro de una clase y quién tiene permiso para hacerlo.

Estos modificadores de acceso te permiten controlar la visibilidad y accesibilidad de los miembros de tus clases, lo que es fundamental para mantener la encapsulación y la seguridad en tu código Java.

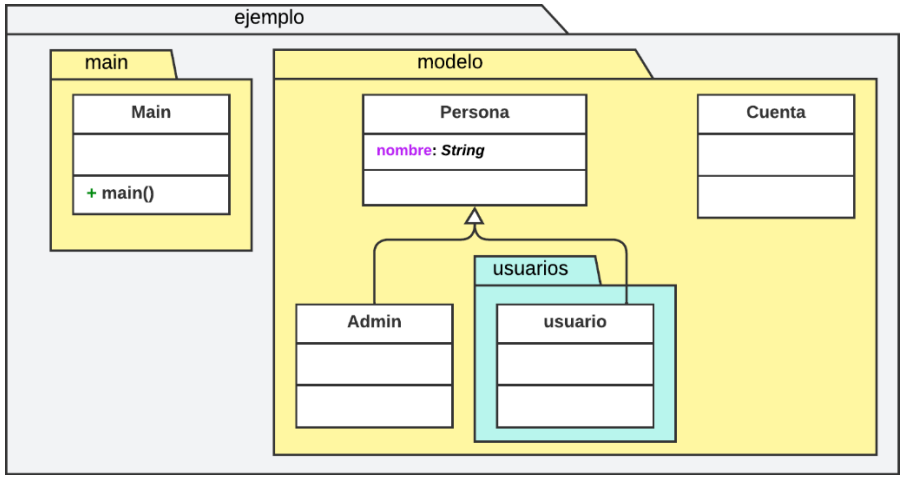
2.2. Tipos de modificadores de acceso

Existen cuatro tipos de modificadores de acceso en Java: **public** (publico), **protected** (protegido), **default** (modificador por defecto) y **private** (privado).

La accesibilidad de los elementos según su modificador de acceso puede verse mediante la siguiente tabla:

		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Public	+	✓	✓	✓	✓	✓
Protected	#	✓	✓	✗	✓	✗
Default	~	✓	✓	✓	✗	✗
Private	-	✓	✗	✗	✗	✗

Vamos a utilizar el siguiente diagrama UML para analizar los diferentes niveles de acceso de cada modificador:



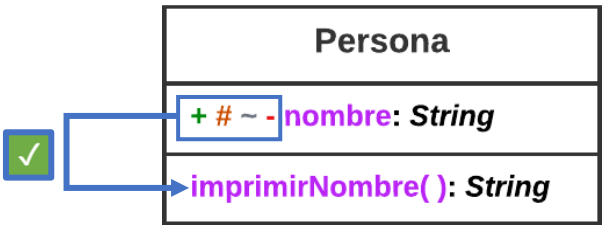
2.2.1. Explicación Niveles de Acceso

Nivel de acceso: Misma clase

		Misma Clase
Public	+	✓
Protected	#	✓
Default	~	✓
Private	-	✓

En la tabla podremos apreciar que, sin importar el modificador de acceso del elemento, siempre se tendrá acceso a ese elemento desde la misma clase en la que está declarado.

Por ejemplo: Tengo un método (o función) llamado “imprimirNombre” declarado en la clase Persona que accede al atributo “nombre” para imprimir por consola el valor que tenga guardado. Sin importar que tipo de modificador de acceso tiene “nombre” cualquier método declarado en esa misma clase tendrá acceso al mismo porque están ambos (atributo y método) en la misma clase.

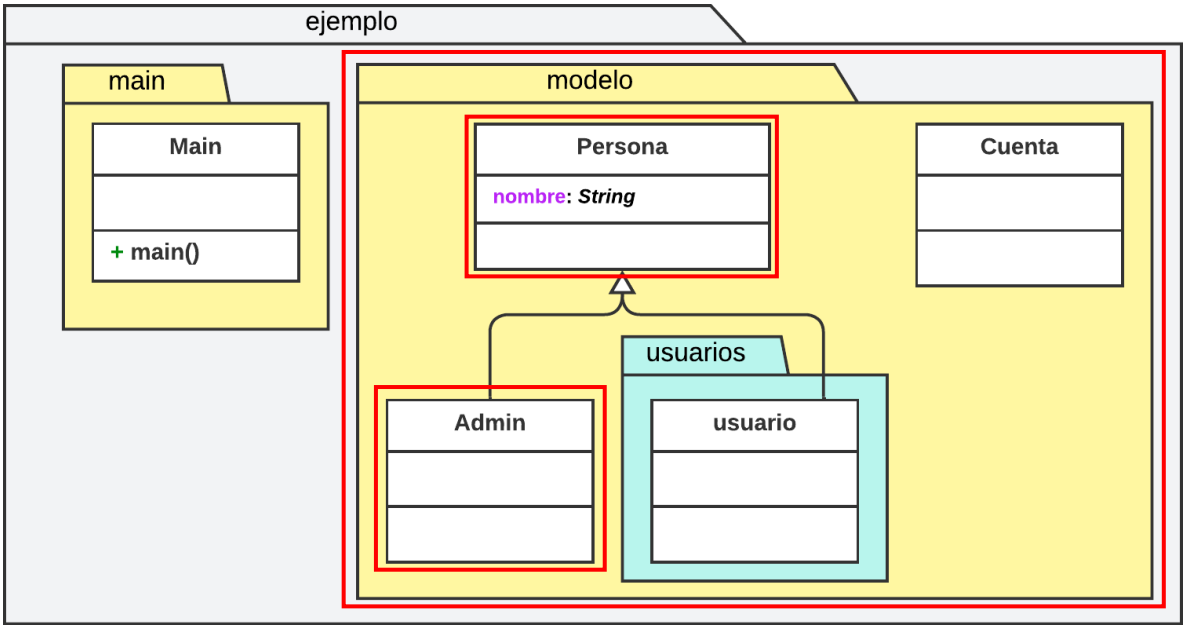


Por esta razón no vamos a analizar este caso para cada uno de los modificadores de acceso ya que independientemente del modificador que se use, el acceso a nivel de la misma clase siempre tendrá el mismo comportamiento.

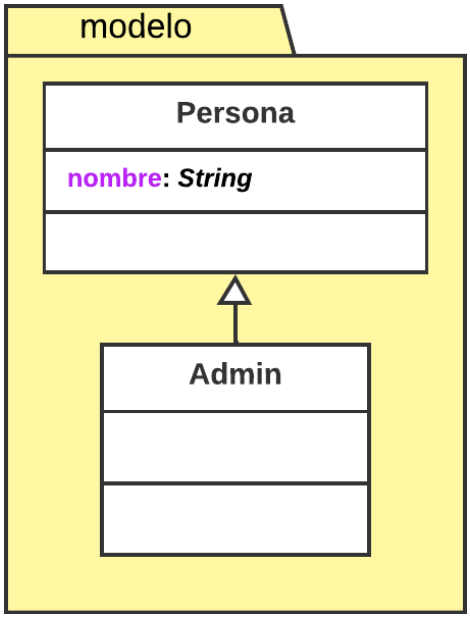
Nivel de acceso: Subclase/Herencia en el mismo paquete

Este nivel de acceso hace referencia en el caso de que un elemento de una clase padre o superclase y su clase hija o subclase se encuentren exactamente dentro del mismo paquete.

La clase persona se encuentra en el paquete "modelo". A su vez, "Persona" tiene una subclase que se encuentra en ese mismo paquete "modelo" llamada "Admin".



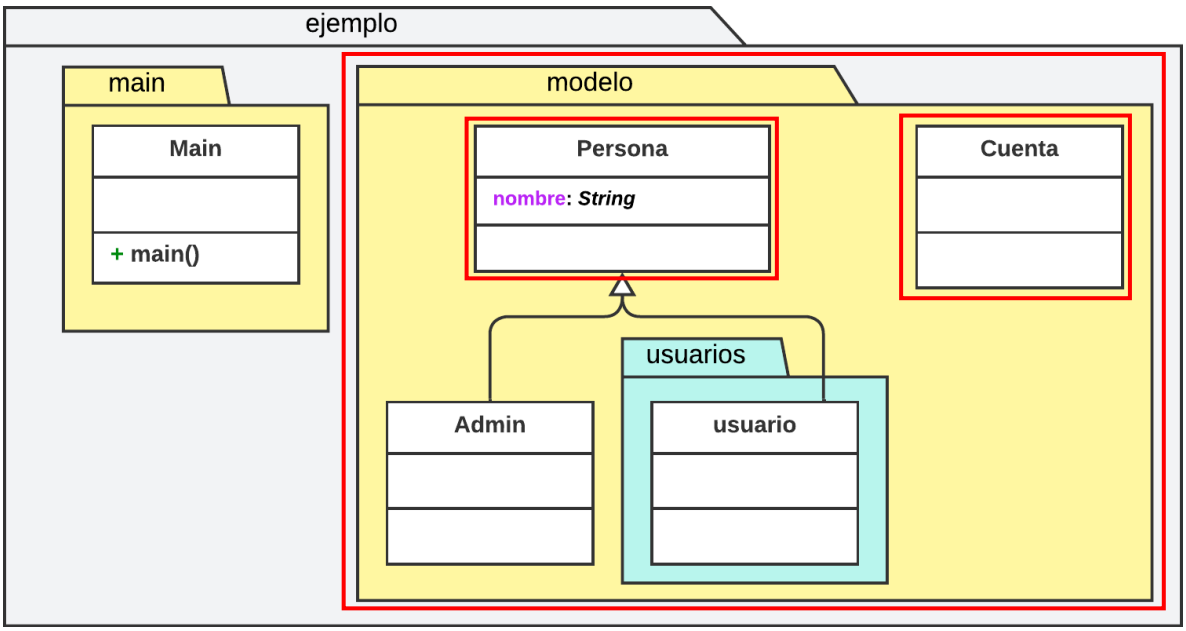
Viendo en el diagrama UML solo a esas clases y paquete tendríamos:



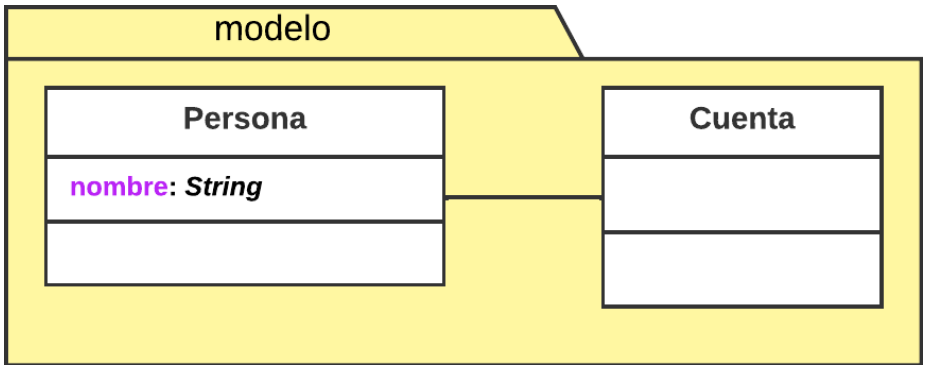
Nivel de acceso: Otra clase en el mismo paquete

Hace referencia cuando un elemento de una clase y otra clase (que no existe herencia entre ellas) se encuentran dentro de exactamente el mismo paquete.

La clase "Persona" se encuentra en el mismo paquete que la clase "Cuenta". Estas dos clases no tienen una relación de herencia.



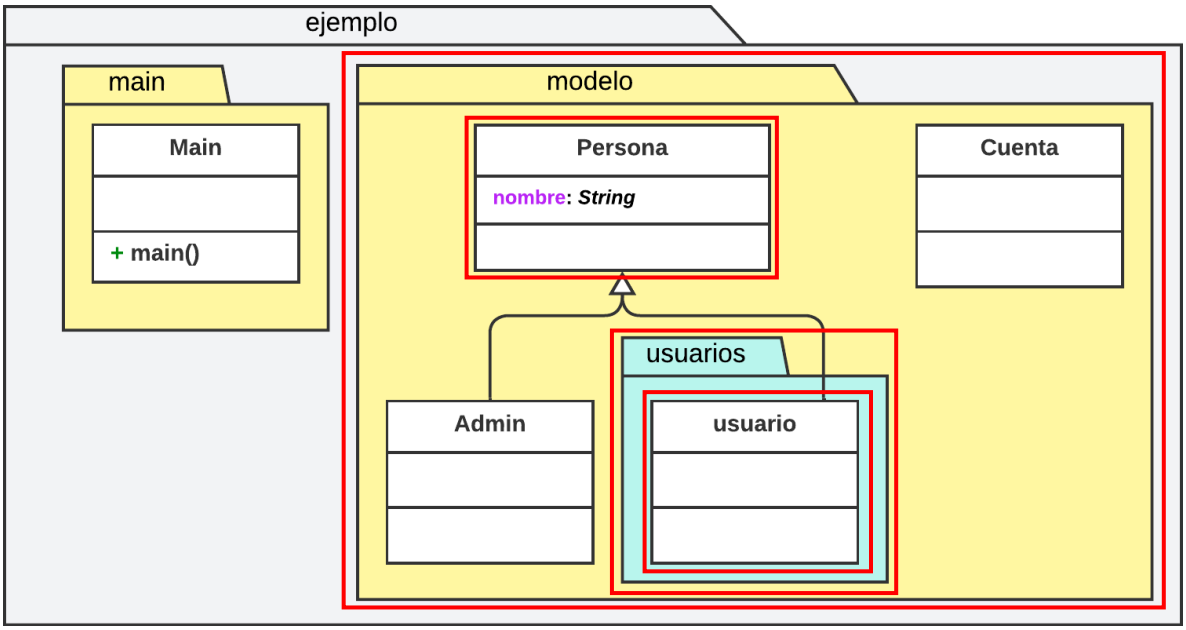
Cuando dos clases se relacionan entre sí a través de objetos, esto se diagrama mediante el uso de una línea contigua sin dirección que une las dos clases. Tomando solo estas dos clases podríamos verlo en UML asi:



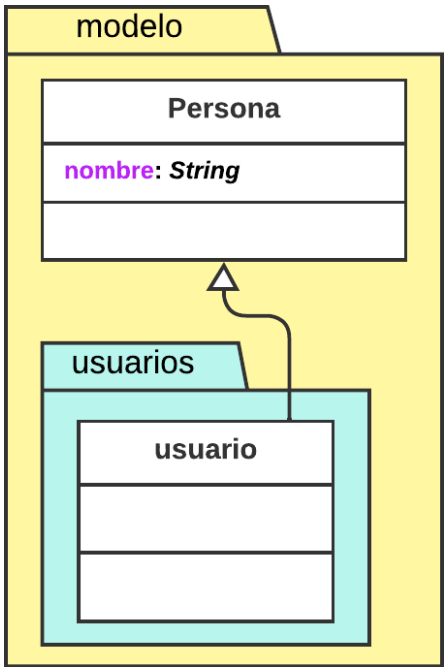
Nivel de acceso: Subclase/Herencia en otro paquete

Hace referencia cuando un elemento de la clase padre se encuentra en un paquete y su clase hija se encuentra en otro paquete distinto.

La clase "Persona" se encuentra dentro del paquete "modelo". La clase hija "usuario" se encuentra dentro del paquete "usuario", por lo que ambas clases a pesar de tener una relación de herencia no se encuentran dentro del mismo paquete.



Solo mostrando los paquetes "modelo" y "usuarios" con ambas clases tendríamos en el diagrama UML:



Nivel de acceso: Otra clase en otro paquete

Hace referencia cuando un elemento de una clase se encuentra en un paquete y otra clase (que no tiene herencia con respecto a la primera clase) se encuentra en un paquete distinto.

La clase "Persona" se encuentra en el paquete "modelo". La clase "Main" se encuentra dentro del paquete "main". Entre estas dos clases no existe una relación de herencia. Ambas clases se encuentran en paquetes distintos.

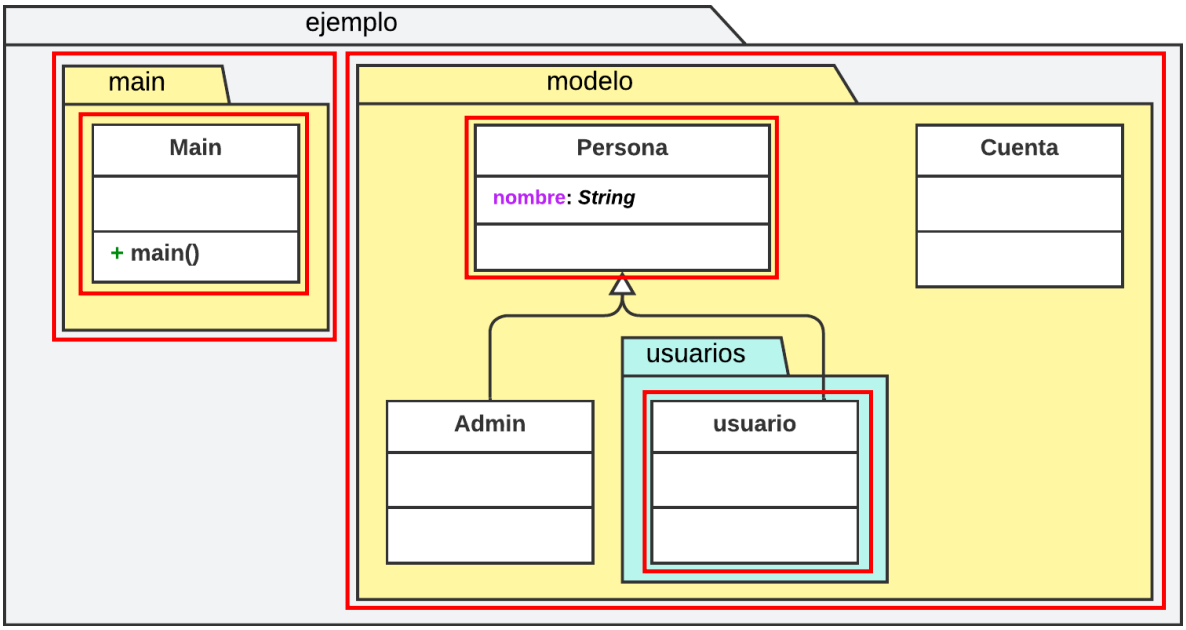
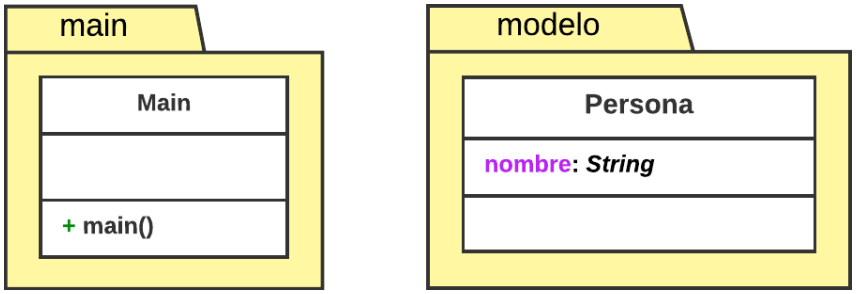


Diagrama UML solo con estos dos paquetes y clases:



2.2.2. Modificador de acceso Public

> *El miembro es accesible desde cualquier clase u objeto en cualquier paquete de nuestro proyecto.*

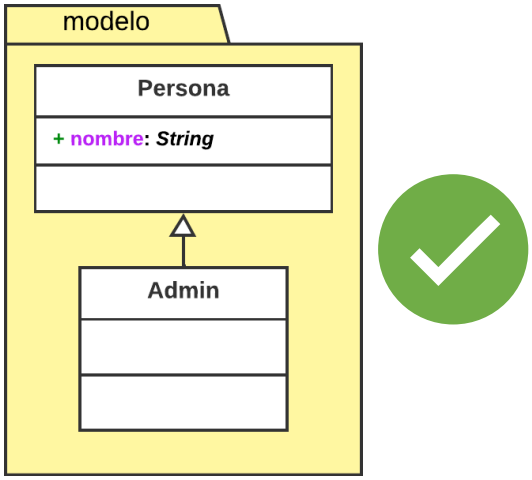
		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Public	+	✓	✓	✓	✓	✓

Cuando utilizamos el modificador de acceso public sobre un elemento de una clase, en cualquier parte de nuestro proyecto podremos acceder a él.

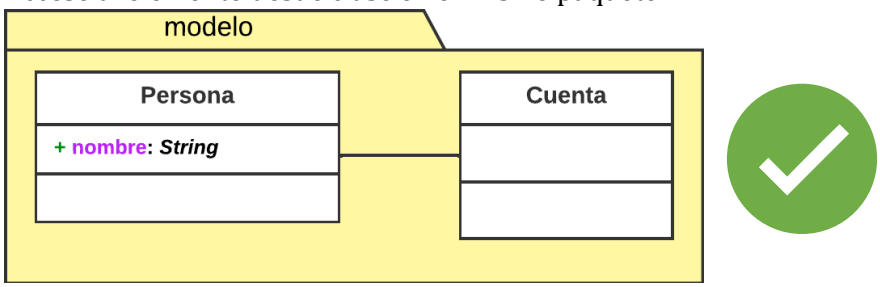
Por esta razón, el método main siempre se declara como "public" al igual que las clases. Si queremos ejecutar un programa necesitamos ejecutar el método main asi que no tendría sentido hacer que este método de "arranque" este oculto de alguna manera. Lo mismo pasa con las clases, lo que queremos lograr es construir un programa en base a los objetos de las clases asi que tampoco tendría mucho sentido ocultar las clases porque no podría existir comunicación entre ellas.

Si declaramos nuestro atributo como public podemos tener:

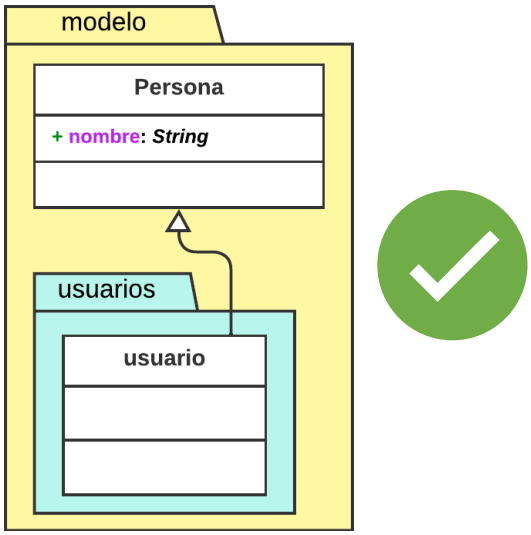
- ✓ Acceso al elemento desde su misma clase
- ✓ Acceso al elemento desde su subclase en el mismo paquete



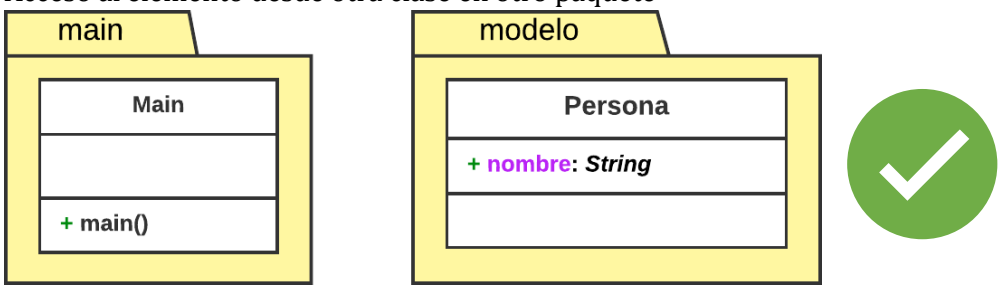
- ✓ Acceso al elemento desde clase en el mismo paquete



✓ Acceso al elemento desde su subclase en otro paquete



✓ Acceso al elemento desde otra clase en otro paquete



2.2.3. Modificador de acceso protected

El miembro es accesible desde la misma clase y clases que sean subclases (heredadas) de esa clase. Los miembros protected no son accesibles desde clases que no sean subclases, incluso si están en el mismo paquete.

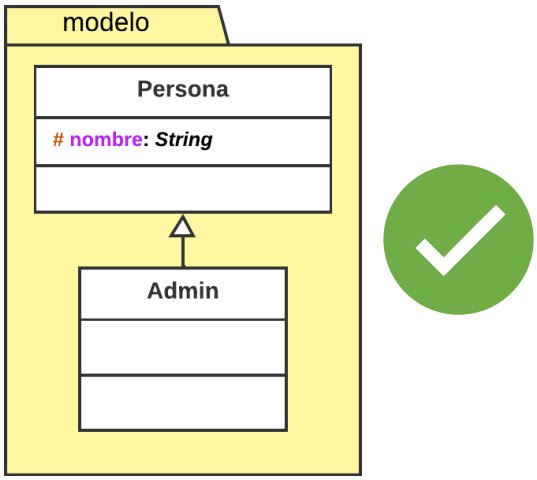
		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Protected	#	✓	✓	✗	✓	✗

Cuando utilizamos el modificador de acceso protected ya empezamos a darle protección a los elementos de una clase.

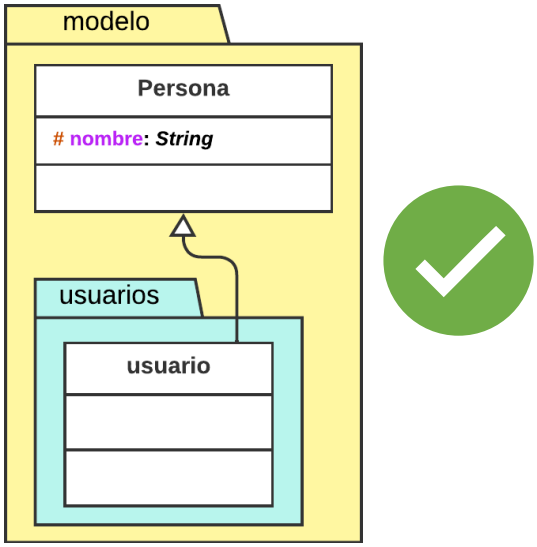
Protected tiene varios usos en la programación java pero mayormente es utilizado para encapsular y proteger los datos de los elementos de una clase pero aun darle permiso a que las clases hijas (subclases) puedan acceder a los elementos de su clase padre (superclase) en la herencia.

Si declaramos nuestro atributo como protected podemos tener:

- ✓ Acceso al elemento desde la misma clase.
- ✓ Acceso al elemento desde una subclase en el mismo paquete.

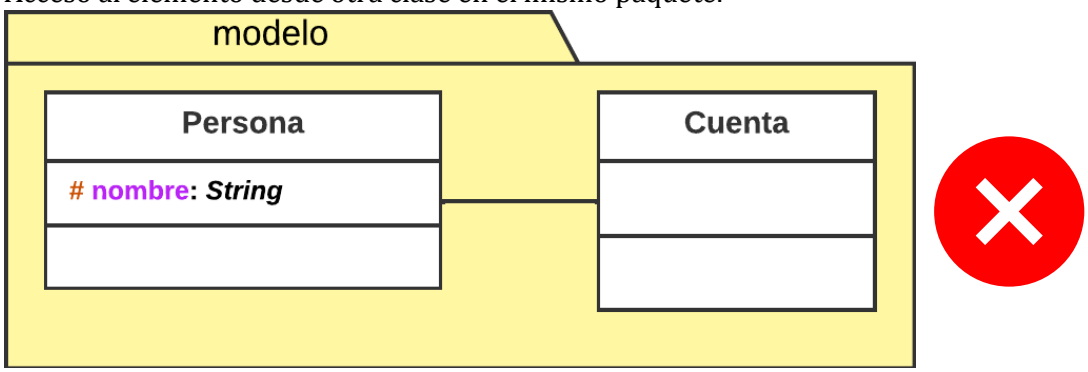


- ✓ Acceso al elemento desde una subclase en otro paquete.

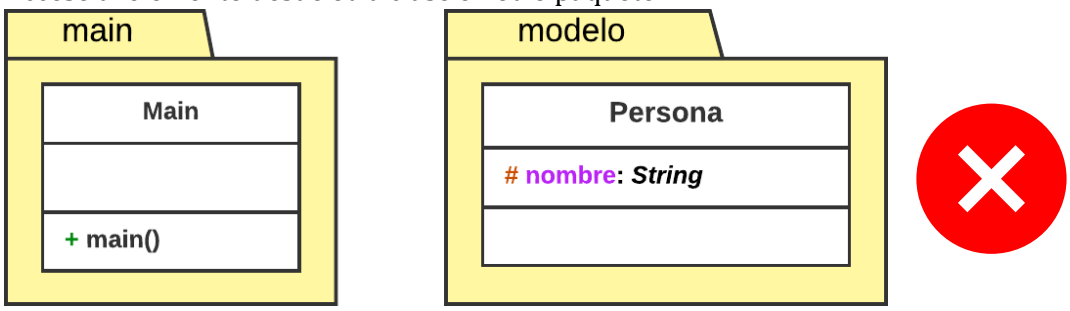


Si declaramos nuestro atributo como protected **NO** podemos tener:

- × Acceso al elemento desde otra clase en el mismo paquete.



- ✗ Acceso al elemento desde otra clase en otro paquete.



2.2.4. Modificador de acceso default

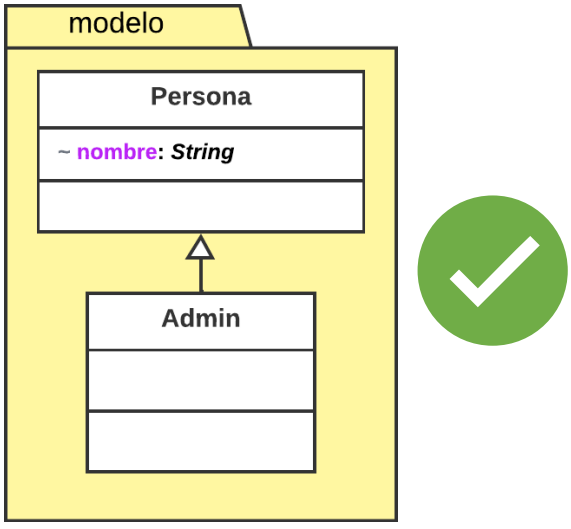
Si no se especifica un modificador de acceso (es decir, no se usa public, protected o private), el miembro es accesible solo dentro del mismo paquete. No es accesible desde clases fuera del paquete.

		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Default	~	✓	✓	✓	✗	✗

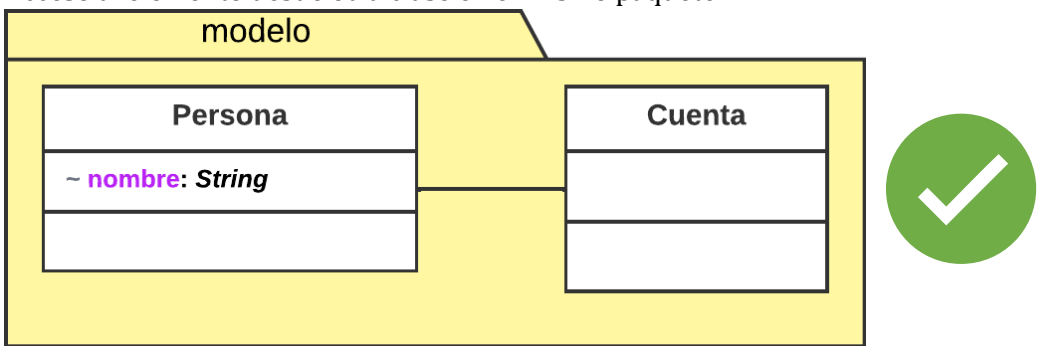
Cuando NO utilizamos ningún modificador de acceso, Java aplicará el modificador de acceso "default" o por defecto. Su principal función es de ocultar toda la información de los elementos para las clases o subclases que no estén dentro de la misma clase donde el elemento se encuentra declarado.

Si declaramos nuestro atributo como default podemos tener:

- ✓ Acceso al elemento desde la misma clase.
- ✓ Acceso al elemento desde una subclase en el mismo paquete.

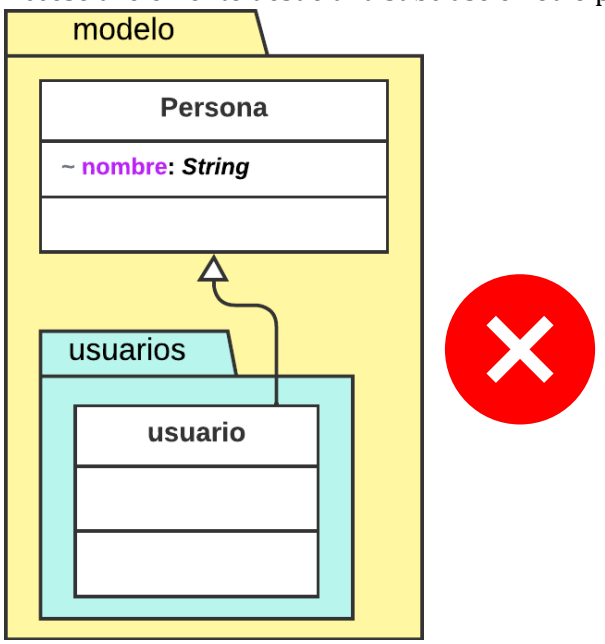


✓ Acceso al elemento desde otra clase en el mismo paquete.

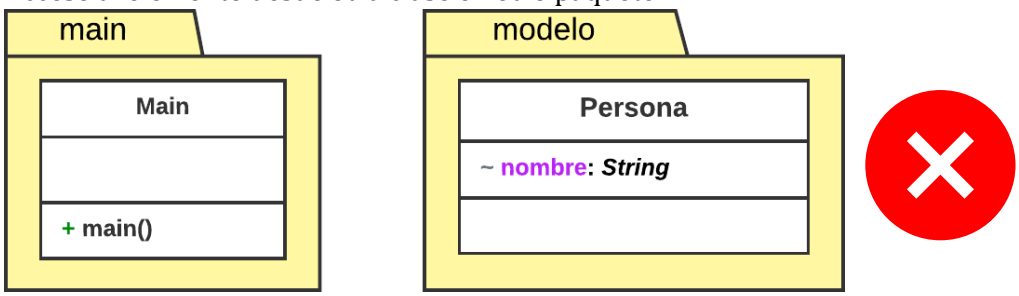


Si declaramos nuestro atributo como default **NO** podemos tener:

× Acceso al elemento desde una subclase en otro paquete.



× Acceso al elemento desde otra clase en otro paquete.



2.2.4. Modificador de acceso private

> *El miembro es accesible solo desde la misma clase. No es accesible desde clases externas, incluso si están en el mismo paquete.*

		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Private	-	✓	X	X	X	X

El modificador de acceso private es el modificador más restrictivo de todos. Solo se puede acceder a los elementos desde su propia clase y de ningún otro lugar.

Junto con el modificador de acceso "public", es uno de los modificadores de acceso más utilizados. La razón detrás de esto es que el uso de elementos "private" es una característica fundamental en el principio de encapsulamiento en la programación orientada a objetos java.

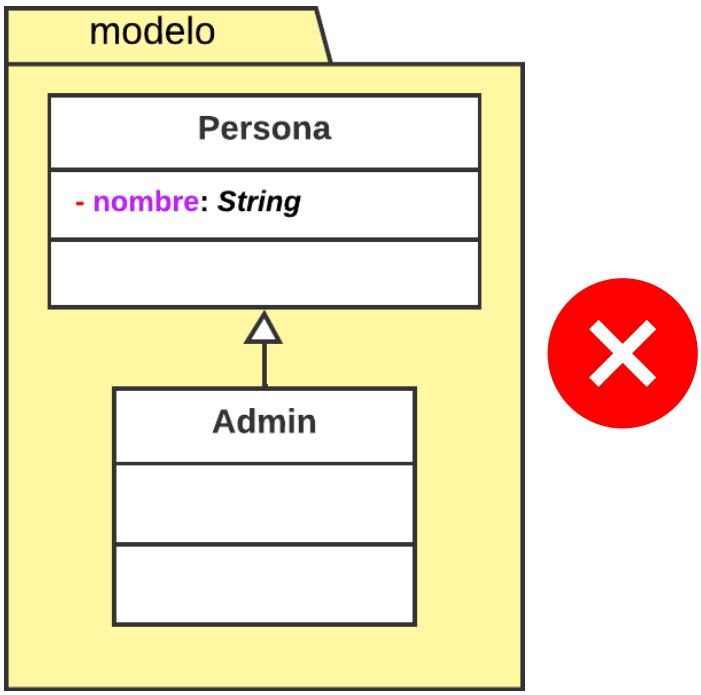
En el capítulo siguiente, veremos cómo se emplea este modificador de acceso en la estructura básica de una clase y cómo funciona el encapsulamiento de los datos de una clase.

Si declaramos nuestro atributo como private podremos:

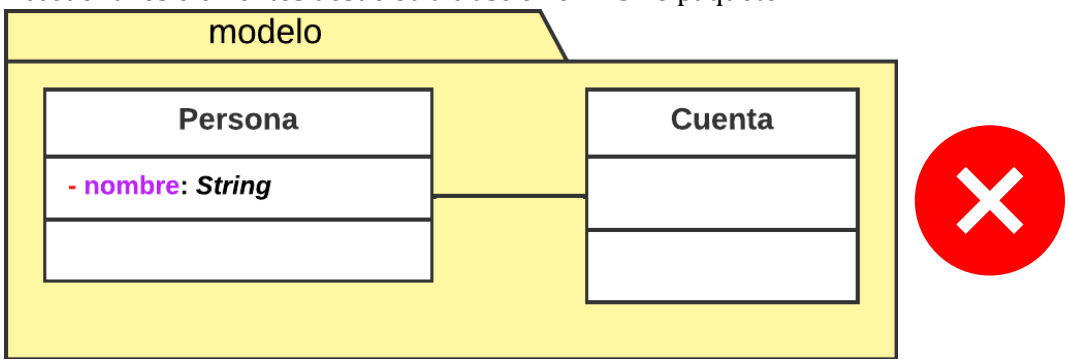
- ✓ Acceder a los elementos desde su misma clase

Si declaramos nuestro atributo como private **NO** podremos:

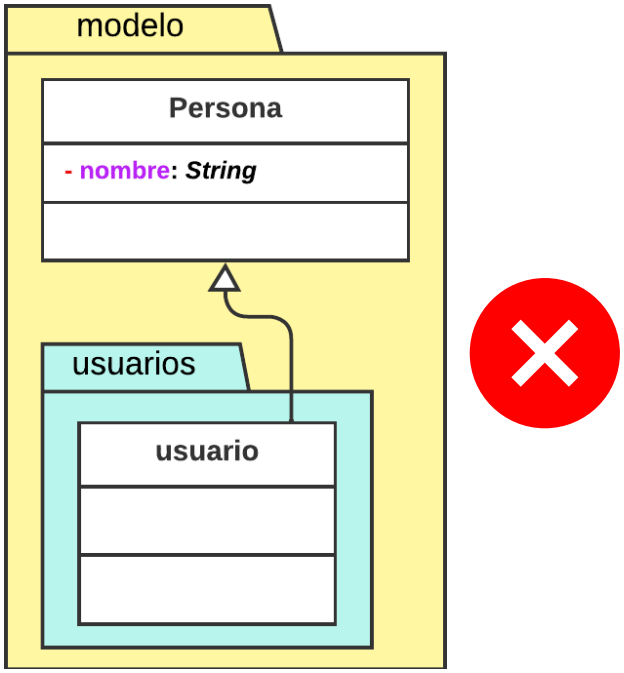
- ✓ Acceder a los elementos desde una subclase en el mismo paquete.



✓ Acceder a los elementos desde otra clase en el mismo paquete.



✓ Acceder a los elementos desde una subclase en otro paquete.



✓ Acceder a los elementos desde otra clase en otro paquete.