Índice de contenido

1. Paquetes en Java

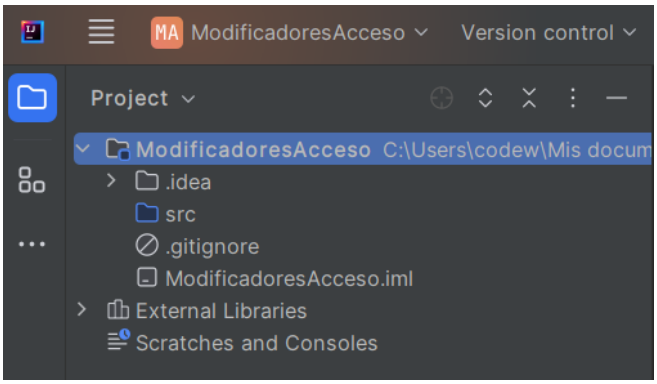
En Java, un paquete es un mecanismo utilizado para organizar y agrupar clases relacionadas. Los paquetes proporcionan un espacio de nombres para las clases, lo que ayuda a evitar conflictos de nombres y facilita la organización del código.

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.

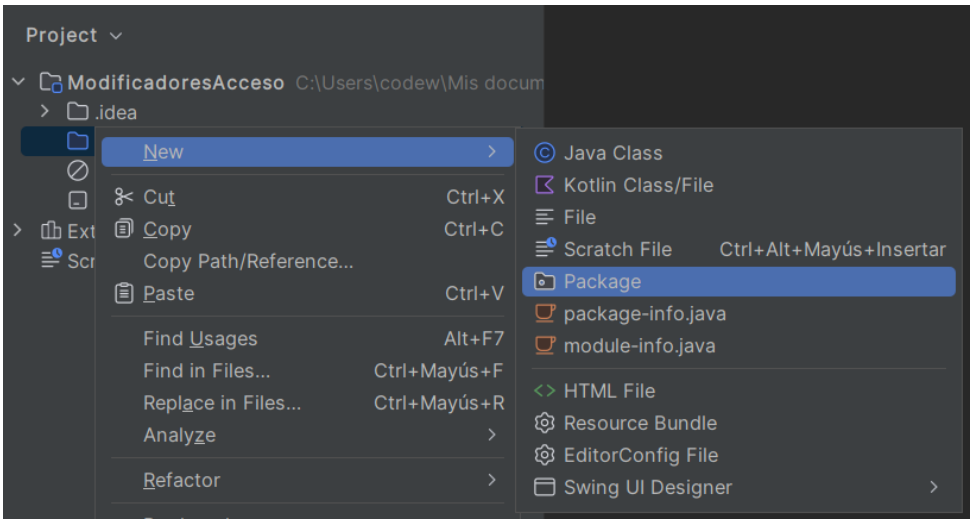
En programas mas complejos y empresariales pueden existir cientos de clases en un mismo proyecto por lo que seria demasiado complicado tener todas las clases en “src” de nuestro proyecto.

1.1. Ejemplo: Creación de un paquete desde IntelliJ

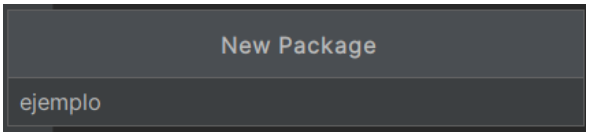
Creamos un nuevo proyecto en IntelliJ llamado “ModificadoresAcceso”.



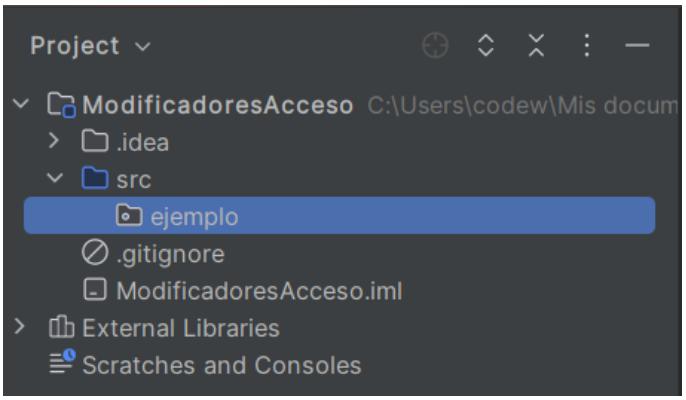
Damos click derecho sobre la carpeta “src” y seleccionamos New -> Package



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.

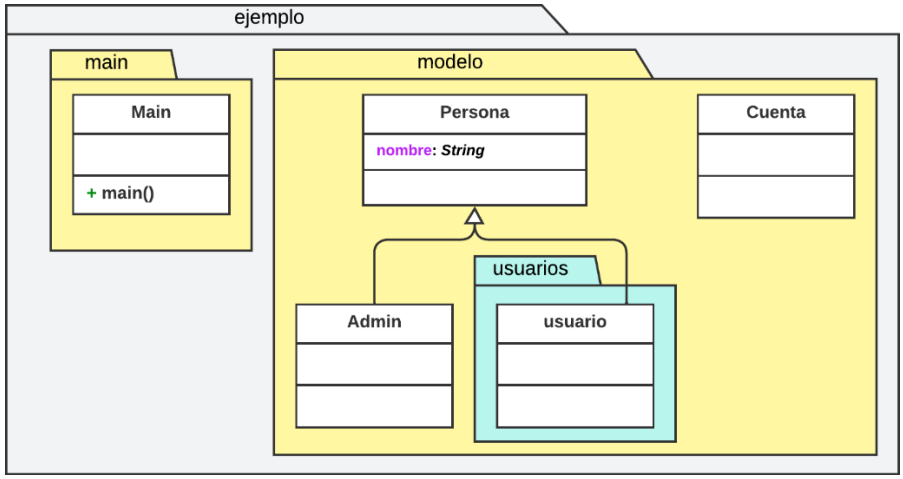
2.2. Tipos de modificadores de acceso

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:



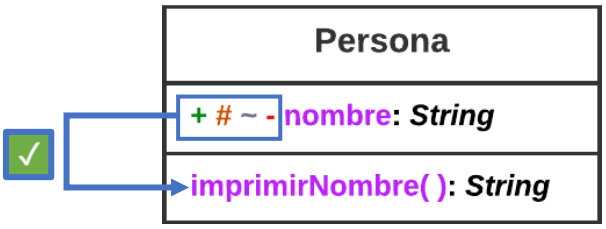
2.2.1. Explicación Niveles de Acceso

Nivel de acceso: Misma clase

		Nivel de acceso
		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.



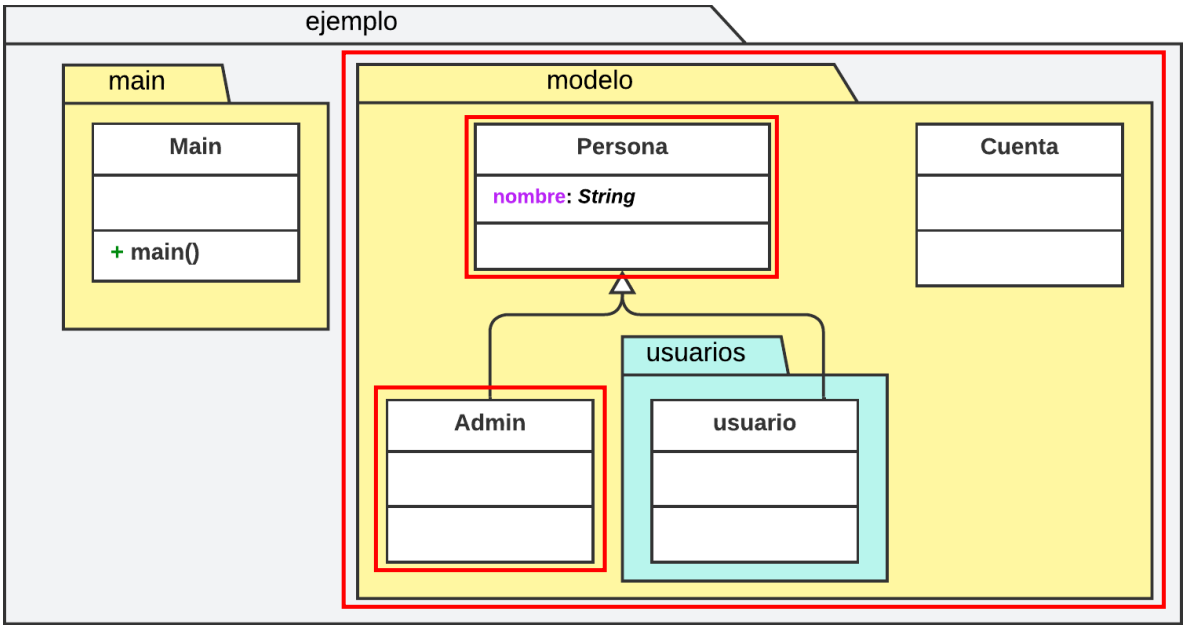
Por esta razón no vamos a analizar este caso para cada uno de los modificadores de acceso ya que independientemente del modificador que se use, el acceso a nivel de la misma clase siempre tendrá el mismo comportamiento.

Nivel de acceso: Subclase/Herencia en el mismo paquete

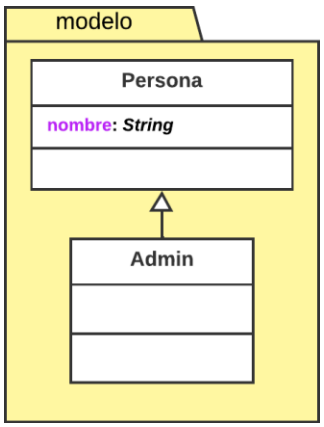
		Nivel de acceso
		Subclase/Herencia en el mismo paquete
Public	+	✓
Protected	#	✓
Default	~	✓
Private	-	✗

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:

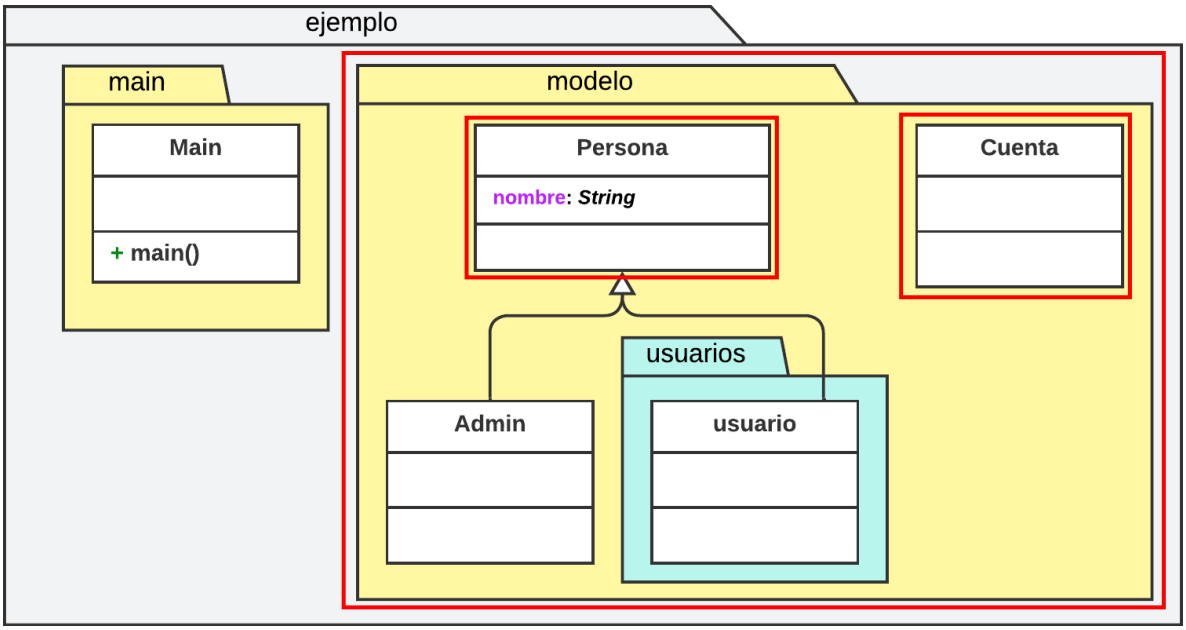


Nivel de acceso: Otra clase en el mismo paquete

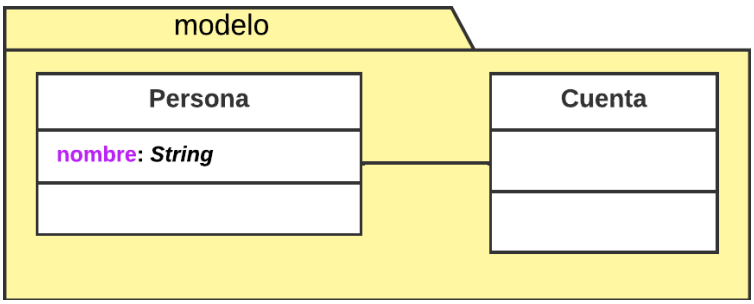
		Nivel de acceso
		Otra clase en el mismo paquete
Public	+	✓
Protected	#	✗
Default	~	✓
Private	-	✗

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:

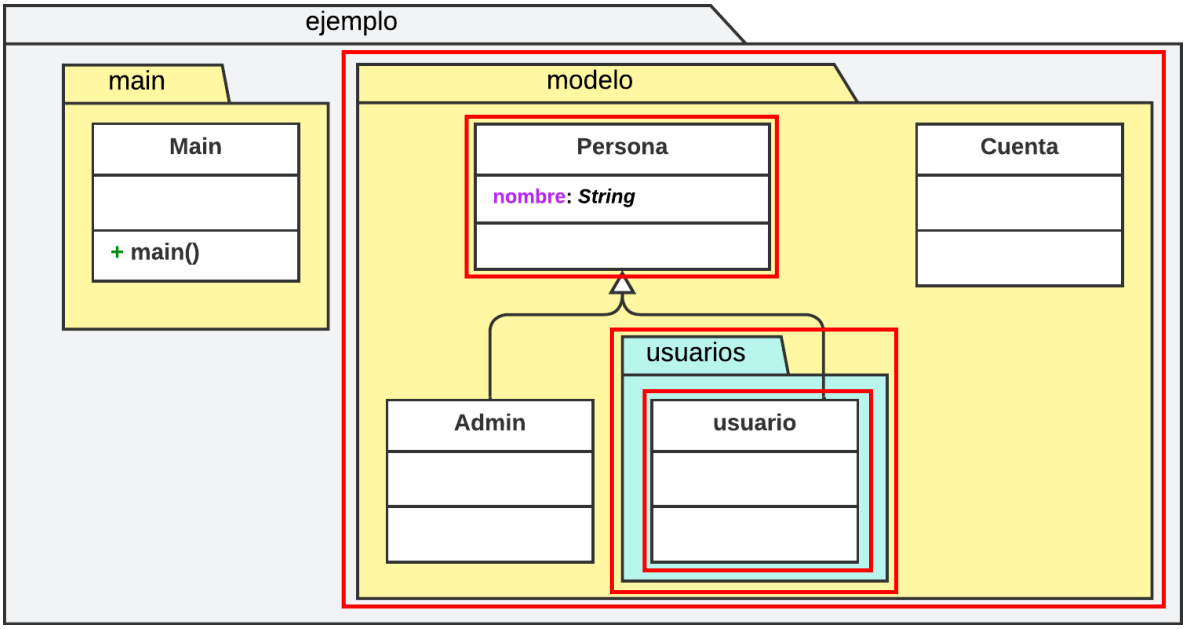


Nivel de acceso: Subclase/Herencia en otro paquete

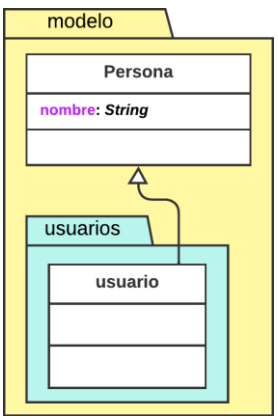
		Nivel de acceso Subclase/Herencia en otro paquete
Public	+	✓
Protected	#	✓
Default	~	✗
Private	-	✗

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

		Nivel de acceso
		Otra clase en otro paquete
Public	+	✓
Protected	#	✗
Default	~	✗
Private	-	✗

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.

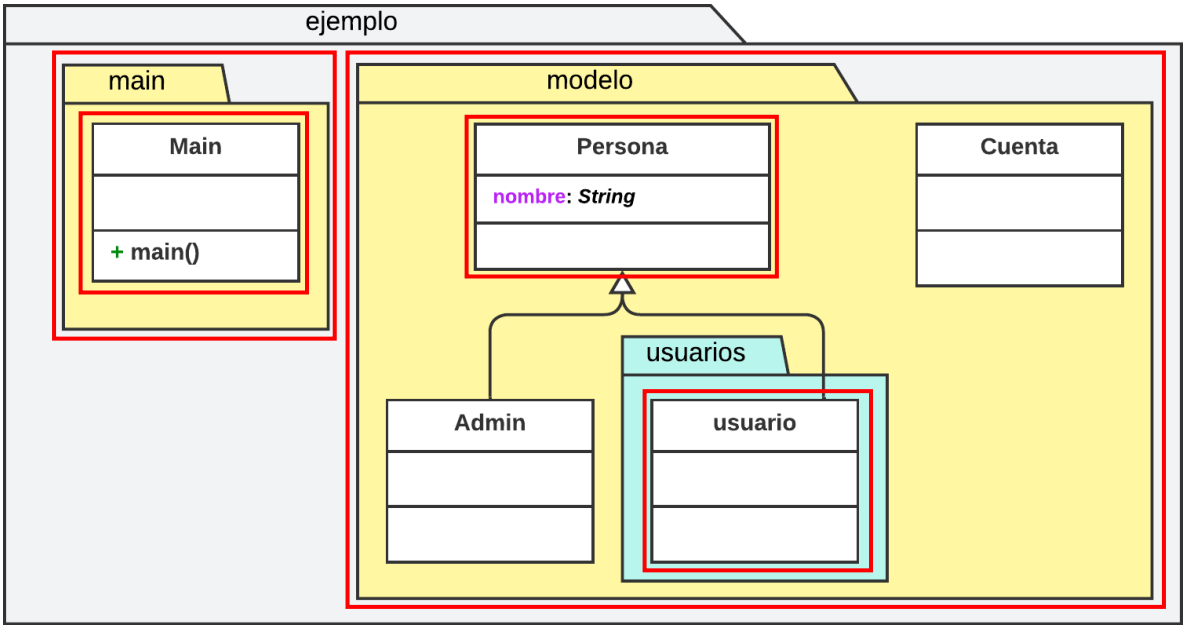
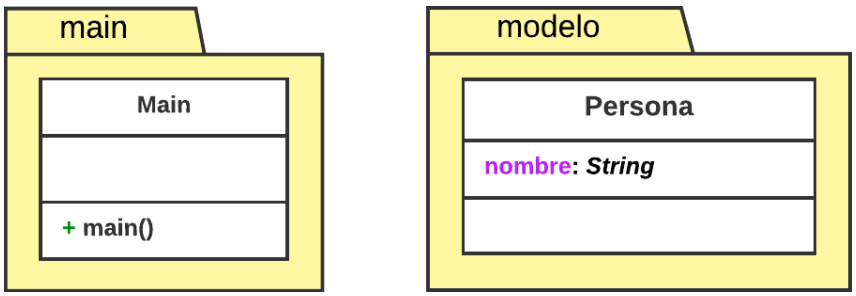


Diagrama UML solo con estos dos paquetes y clases:



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:

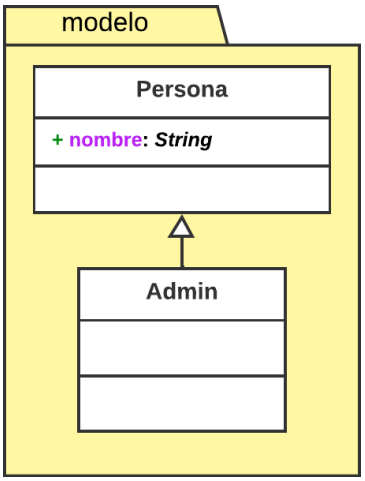
- ✓ Acceso al elemento desde su misma clase

Persona.java

```
package com.codewithomar.modificadores.modelo;

//Clase POJO
public class Persona {
    //Declaración de atributos
    public String nombre;

    //Constructor vacío
    public Persona(){
        this.nombre = "Omar";
    }
    //[...]
}
```

Tenemos declarado al atributo nombre con acceso "public", por lo que puede ser llamado desde otro elemento de la misma clase, en este caso el método constructor Persona().

✓ Acceso al elemento desde su subclase en el mismo paquete



Persona.java Admin.java

```
1  package com.codewithomar.modificadores.modelo;
2
3  public class Admin extends Persona{
4      //Declaración de atributos
5      private String departamento;
6
7      //Constructor vacío
8      public Admin(){
9          super.nombre = "Omar";
10         this.departamento = "Desarrollo";
11     }
12 }
```

La herencia entre dos clases se declara utilizando la palabra clave *extends*.

En la línea 3 se indica que Admin extiende (hereda) de Persona.

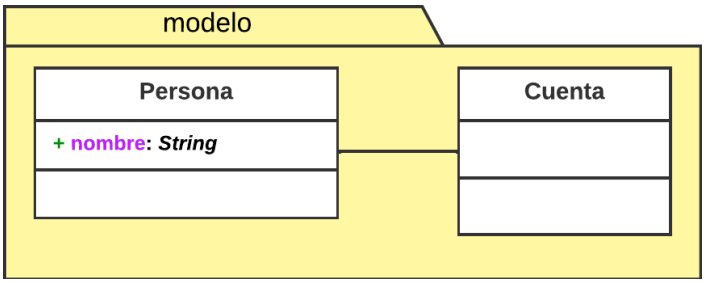
Cuando una clase es hijo de otra clase, **hereda todos sus atributos y métodos**.

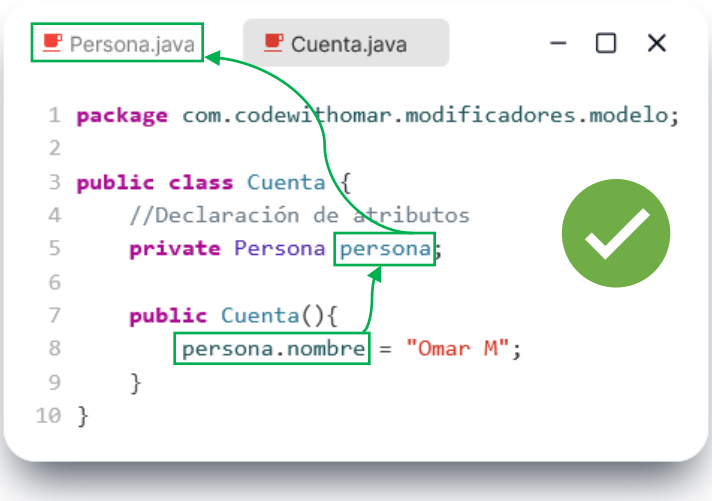
Para hacer el llamado de esos atributos o métodos se utiliza la palabra clave *super*.

En la línea 9, la clase Admin hace un llamado directo al atributo nombre de su clase padre Persona. Como persona es de tipo **public**, su subclase en el mismo paquete puede ser accedido directamente.

La herencia es un tema avanzado que veremos en otra clase.

✓ Acceso al elemento desde clase en el mismo paquete





La clase Cuenta y la clase Persona se encuentran dentro del mismo paquete "modelo". No existe una relación de herencia entre ellas.

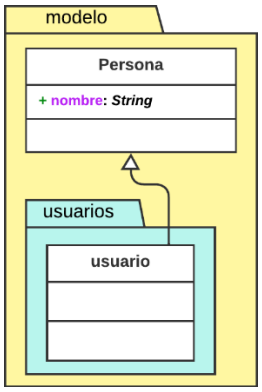
En la clase Cuenta se declara un objeto de la clase Persona. Al crearse un objeto dentro de otra clase, **esta clase podrá llamar a los elementos declarados en la clase objeto**.

Como el atributo "nombre" está declarado como ***public***, la clase Cuenta **puede** llamar al atributo "nombre" del objeto "persona" de la clase "Persona" **directamente**.

Siguiendo los lineamientos de la Encapsulación en Programación Orientada a Objetos, esto NO se debería permitir. Recordemos que el principio de encapsulación nos dice que los atributos deben ser privados para que las clases externas no puedan acceder al contenido de los atributos directamente; si desean acceder a los atributos, deberán hacerlo mediante el uso de los métodos getters y setters del atributo.

Este ejemplo se mostró por mera demostración y para enseñar que en efecto si se puede acceder

✓ Acceso al elemento desde su subclase en otro paquete



Persona.java Usuario.java

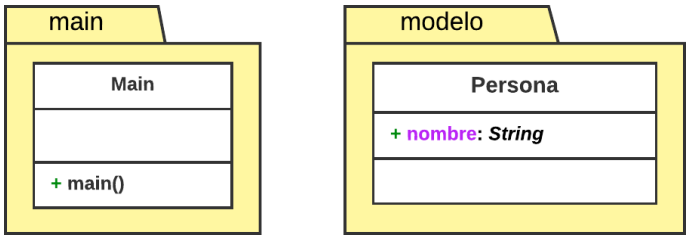
```
1  package com.codewithomar.modificadores.modelo.usarios;
2
3  import com.codewithomar.modificadores.modelo.Persona;
4
5  public class Usuario extends Persona {
6      private String nombreUsuario;
7
8      public Usuario(){
9          super.nombre = "Omar";
10         this.nombreUsuario = "codewithomarm";
11     }
12 }
```

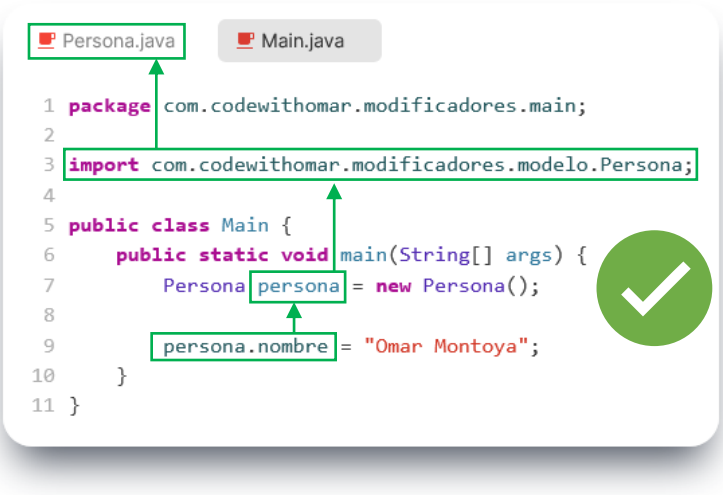
La clase Usuario extiende de la clase Persona y se encuentra en el paquete “usuarios”. El paquete “usuarios” a su vez se encuentra dentro del paquete “modelo” donde la clase Persona también se encuentra.

Como Usuario es hijo de Persona, puede llamar a su atributo “nombre” utilizando la palabra clave “super”. Como el atributo nombre de Persona tiene modificador de acceso “public”, la clase hija desde otro paquete puede llamarlo directamente.

Debido a que Persona se encuentra en otro paquete, debemos hacer el import de la línea 3 para poder acceder al contenido de la clase Persona.

✓ Acceso al elemento desde otra clase en otro paquete





La clase Main se encuentra dentro del paquete “main”. El paquete “main” no forma parte del paquete “modelo” donde se encuentra “Persona”.

Como la clase Main utiliza un objeto de la clase Persona, se necesita un import de la ubicación de la clase Persona en el proyecto (línea 3).

El método main crea un objeto de la clase Persona (Línea 7).

El método main llama directamente al atributo nombre del objeto persona. Esto es permitido debido a que el atributo nombre tiene modificador de acceso público.

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	#	✓	✓	X	✓	X

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.

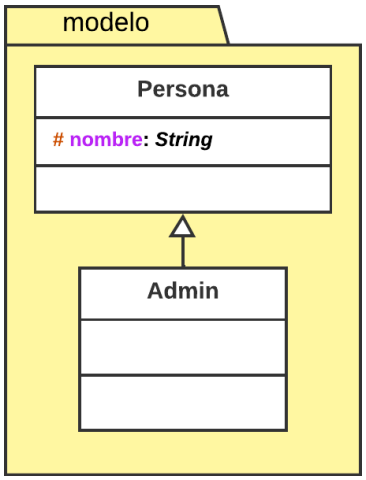
Persona.java

```java
package com.codewithomar.modificadores.modelo;

//Clase POJO
public class Persona {
    //Declaración de atributos
    protected String nombre;

    //Constructor vacío
    public Persona(){
        this.nombre = "Omar";
    }
    //[...]
}
```

Sin importar el modificador de acceso, los elementos de la clase siempre tendrán acceso a otros elementos dentro de la clase.

Utilizando el modificador de acceso protected, el método Constructor puede llamar directamente al atributo nombre y modificarlo.

✓ Acceso al elemento desde una subclase en el mismo paquete.



Persona.java Admin.java

```
1  package com.codewithomar.modificadores.modelo;
2
3  public class Admin extends Persona{
4      //Declaración de atributos
5      private String departamento;
6
7      //Constructor vacío
8      public Admin(){
9          super.nombre = "Omar";
10         this.departamento = "Desarrollo";
11     }
12 }
```

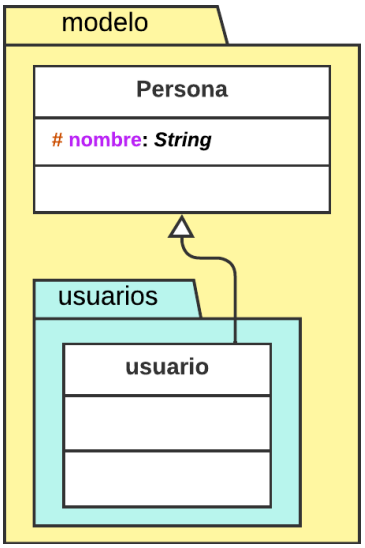
El modificador de acceso protected se utiliza específicamente para el encapsulamiento de datos en la herencia.

El modificador de acceso protected, permite que las clases hijas puedan acceder directamente a los elementos del padre.

En este caso, la clase hija se encuentra dentro del **mismo paquete** que la clase padre.

En la línea 9 se hace el llamado directo al atributo "nombre" de la clase padre Persona para asignarle un valor y es permitido.

✓ Acceso al elemento desde una subclase en otro paquete.



Persona.java Usuario.java

```
1  package com.codewithomar.modificadores.modelo.usarios;
2
3  import com.codewithomar.modificadores.modelo.Persona;
4
5  public class Usuario extends Persona {
6      private String nombreUsuario;
7
8      public Usuario(){
9          super.nombre = "Omar";
10         this.nombreUsuario = "codewithomarm";
11     }
12 }
```

El modificador de acceso protected se utiliza específicamente para el encapsulamiento de datos en la herencia.

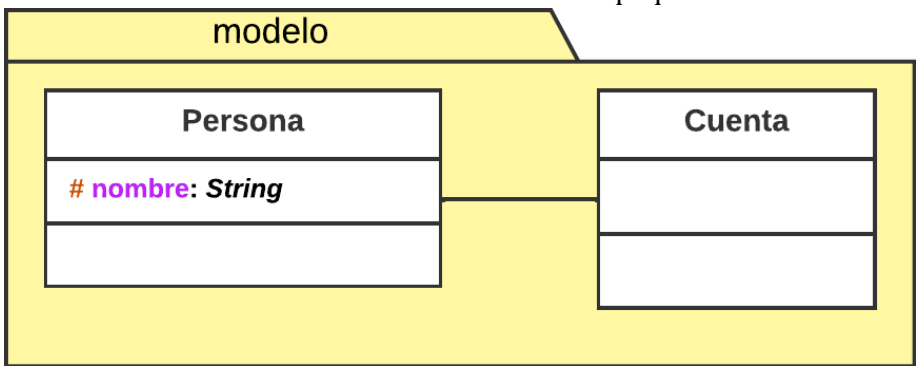
El modificador de acceso protected, permite que las clases hijas puedan acceder directamente a los elementos del padre, **independientemente del paquete en el que se encuentren**.

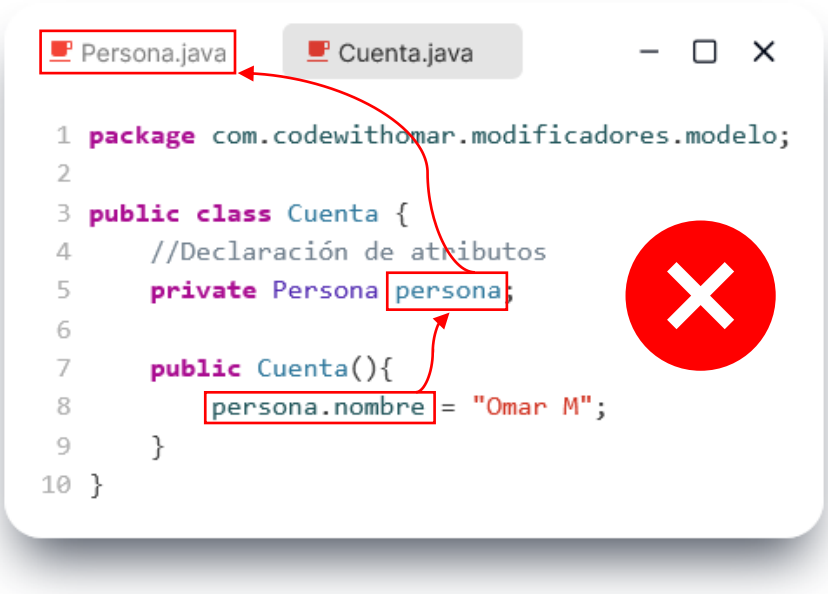
En este caso, la clase hija se encuentra dentro de un **paquete diferente** que la clase padre.

En la línea 9 se hace el llamado directo al atributo "nombre" de la clase padre Persona para asignarle un valor y es permitido debido al modificador de acceso protected.

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

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





La clase Cuenta y la clase Persona se encuentran dentro del mismo paquete "modelo". **No existe una relación de herencia entre ellas.**

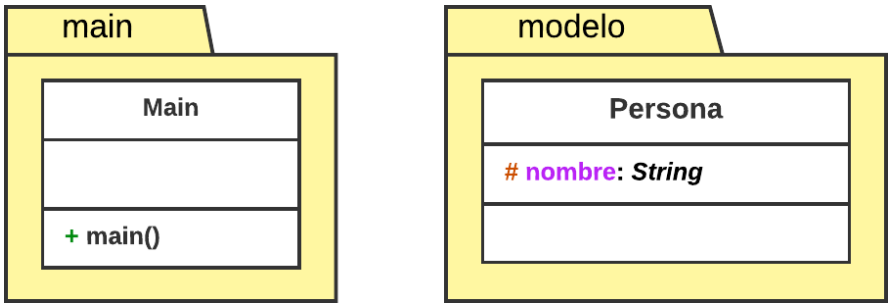
Como el atributo "nombre" está declarado como ***protected***, la clase Cuenta **NO puede** llamar al atributo "nombre" del objeto "persona" de la clase "Persona" **directamente**.

Siguiendo los lineamientos de la Encapsulación en Programación Orientada a Objetos, el modificador de acceso protected solo permitirá el llamado directo de elementos entre las clases hijas y las clases padres; es decir, las clases que tienen relaciones de herencia entre ellas.

Como entre Persona y Cuenta no existe relación de herencia, **no se puede hacer el llamado directo al atributo "nombre" del objeto persona**.

Si se desea modificar el valor de un atributo protected desde una clase en el mismo paquete, se deberá hacer el llamado al método *Setter* del atributo. Si se desea retornar el valor del atributo protected, se utilizará el método *Getter* del atributo. Los métodos Setter y Getter siempre se declaran *public* para que estas acciones puedan ser realizadas.

- Acceso al elemento desde otra clase en otro paquete.



Persona.java Main.java

```
1  package com.codewithomar.modificadores.main;
2
3  import com.codewithomar.modificadores.modelo.Persona;
4
5  public class Main {
6      public static void main(String[] args) {
7          Persona persona = new Persona();
8
9          persona.nombre = "Omar Montoya";
10     }
11 }
```

La clase Main se encuentra dentro del paquete “main”. El paquete “main” no forma parte del paquete “modelo” donde se encuentra “Persona”. Por ende, Persona y Main se encuentran en paquetes distintos y no existe una relación de herencia entre estas dos clases.

Como no existe relación de herencia entre las dos clases, no se puede llamar directamente al atributo desde la clase Main y modificar su valor.

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.

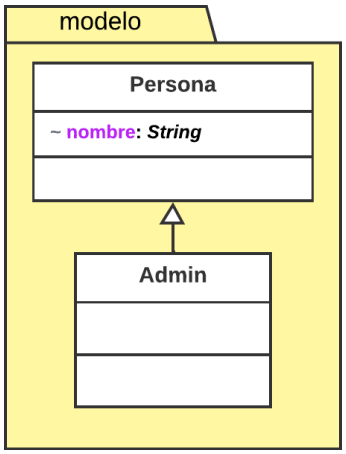
Persona.java

```
1  package com.codewithomar.modificadores.modelo;
2
3  //Clase POJO
4  public class Persona {
5      //Declaración de atributos
6      String nombre;
7
8      //Constructor vacío
9      public Persona(){
10         this.nombre = "Omar";
11     }
12     //[...]
13 }
```

En la línea 6 podemos notar que no está escrito ningún modificador de acceso en la declaración del atributo nombre.

La ausencia de modificador de acceso indica que se utilizará el modificador de acceso protected.

✓ Acceso al elemento desde una subclase en el mismo paquete.



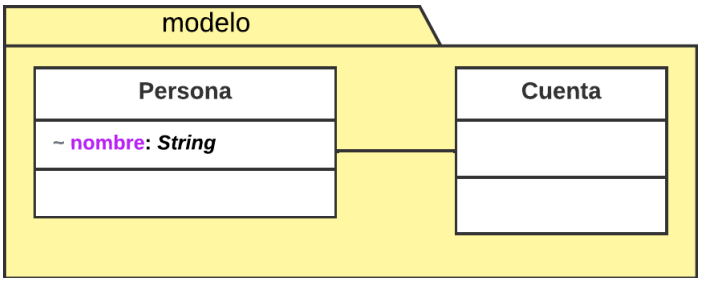
Persona.java Admin.java

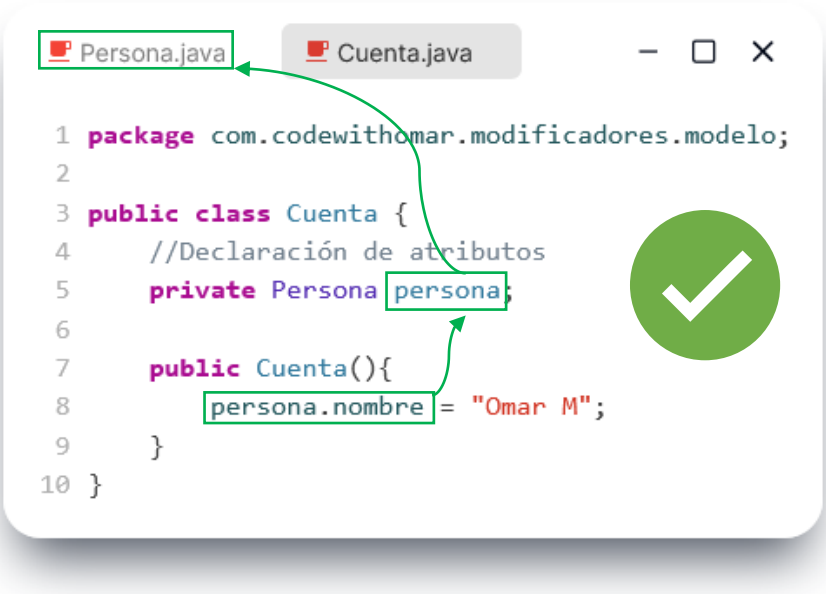
```java
package com.codewithomar.modificadores.modelo;

public class Admin extends Persona{
    //Declaración de atributos
    private String departamento;

    //Constructor vacío
    public Admin(){
        super.nombre = "Omar";
        this.departamento = "Desarrollo";
    }
}
```

Protected permite acceder a los elementos de una clase que se encuentran dentro del mismo paquete.

Como Persona y Admin se encuentran dentro del mismo paquete "modelo" y existe una relación de herencia entre ambos, Admin puede acceder directamente al atributo "nombre" de su superclase (clase padre) mediante la palabra clave "super".

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



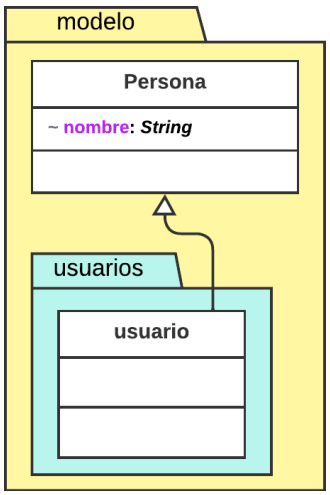


La clase Cuenta y Persona se encuentran dentro del mismo paquete "modelo" y no existe relación de herencia entre ellas.

Como se encuentran dentro del mismo paquete, se permite acceder directamente al atributo "nombre" del objeto persona debido al modificador de acceso "default".

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

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



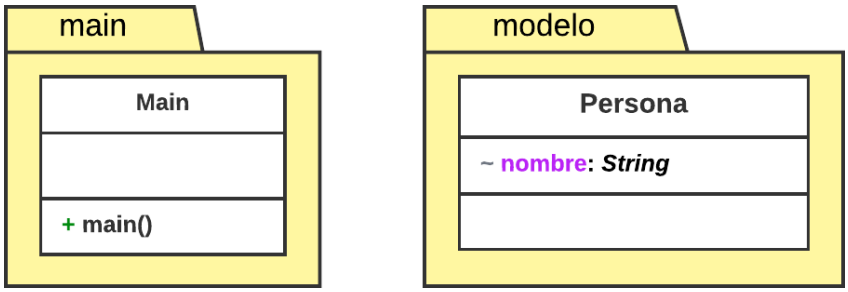
Persona.java Usuario.java

```java
package com.codewithomar.modificadores.modelo.usarios;

import com.codewithomar.modificadores.modelo.Persona;

public class Usuario extends Persona {
    private String nombreUsuario;

    public Usuario(){
        super.nombre = "Omar";
        this.nombreUsuario = "codewithomarm";
    }
}
```

El modificador de acceso default no permite que clases que no estén dentro del mismo paquete puedan acceder a los elementos.

Si bien Persona y Usuario se encuentran dentro del mismo paquete y existe una relación de herencia entre ellos, como usuario se encuentro dentro de un subpaquete, para Java ya no se encuentran dentro del mismo nivel de paquetes por lo que no permite acceder directamente a los elementos de su superclase (clase padre).

- Acceso al elemento desde otra clase en otro paquete.



Persona.java Main.java

```
1  package com.codewithomar.modificadores.main;
2
3  import com.codewithomar.modificadores.modelo.Persona;
4
5  public class Main {
6      public static void main(String[] args) {
7          Persona persona = new Persona();
8
9          persona.nombre = "Omar Montoya";
10     }
11 }
```

La clase Main se encuentra en un paquete completamente aparte al paquete de la clase Persona.

Debido a que el atributo nombre tiene el modificador de acceso protected, no se puede acceder directamente a este elemento ya que no se encuentran dentro del mismo 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

Persona.java

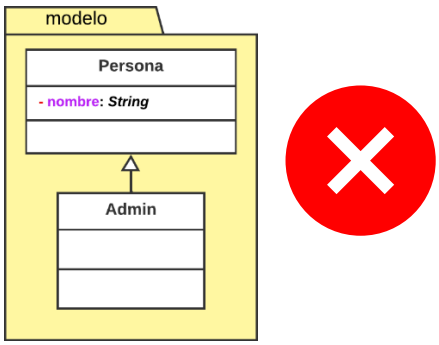
```
1  package com.codewithomar.modificadores.modelo;
2
3  //Clase POJO
4  public class Persona {
5      //Declaración de atributos
6      private String nombre;
7
8      //Constructor vacío
9      public Persona(){
10         this.nombre = "Omar";
11     }
12     //[...]
13 }
```

El modificador de acceso private esta hecho para que solo se pueda acceder al elemento desde la misma clase donde está declarado.

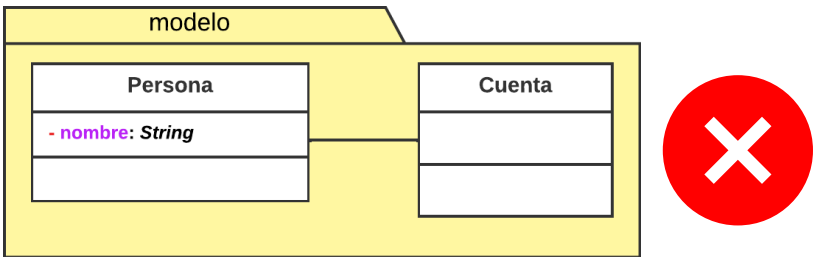
Como el atributo nombre es private, cualquier elemento dentro de esa clase puede llamar directamente al atributo y modificar su contenido o utilizarlo.

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

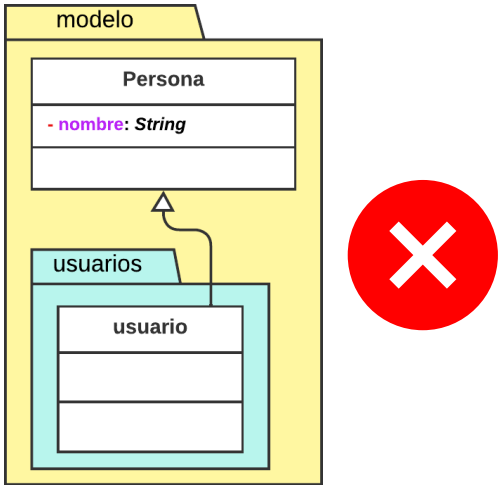
- ✓ Acceder a los elementos desde una subclase en el mismo paquete.



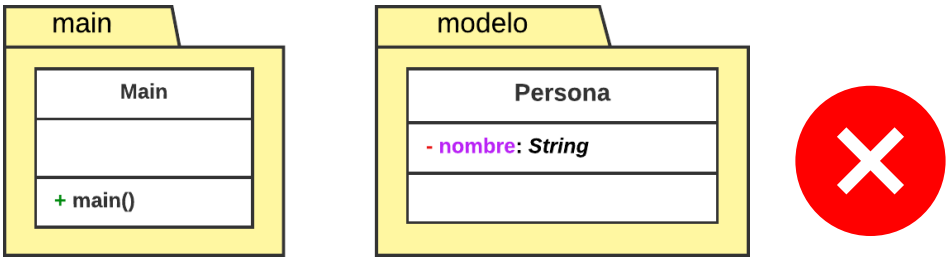
- ✓ Acceder a los elementos desde otra clase en el mismo paquete.



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



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



2.3. ¿Cuál modificador de acceso utilizar?

Ahora que ya vimos todos los posibles usos de cada modificador de acceso y como se comportan en base a la ubicación del paquete de las clases y la herencia entre ellas; probablemente te preguntarás: entonces, ¿Cuál modificador debo utilizar?

La decisión de qué modificador de acceso utilizar puede basarse en:

- Principios de encapsulamiento de POO.
- Composición de nuestro proyecto.

Según Oracle debemos seguir los siguientes lineamientos al asignar modificadores de acceso a nuestros elementos:

1. Utilizar el modificador de acceso más restrictivo que haga sentido para un elemento de una clase.
2. Utilizar el modificador de acceso private a menos que tengamos una razón para utilizar uno distinto.
3. Evitar atributos public excepto para constantes (final / static final).

2.3.1. Cuando usar modificador de acceso public

El modificador de acceso public se deberá usar de la siguiente manera:

- Las clases deberán ser public para poder crear objetos de ellas en otras clases y utilizar los objetos. De lo contrario algunas clases que necesiten utilizar objetos de estas clases podrían no tener acceso a ella. Es la práctica mas común.

Persona.java

```java
package com.codewithomar.modificadores.modelo;

//Clase POJO
public class Persona {
    //[...]
}
```

- Los métodos que son necesarios para manipular los objetos en otras clases deberán ser declarados public. Esto garantiza el acceso a los métodos desde cualquier parte del proyecto.

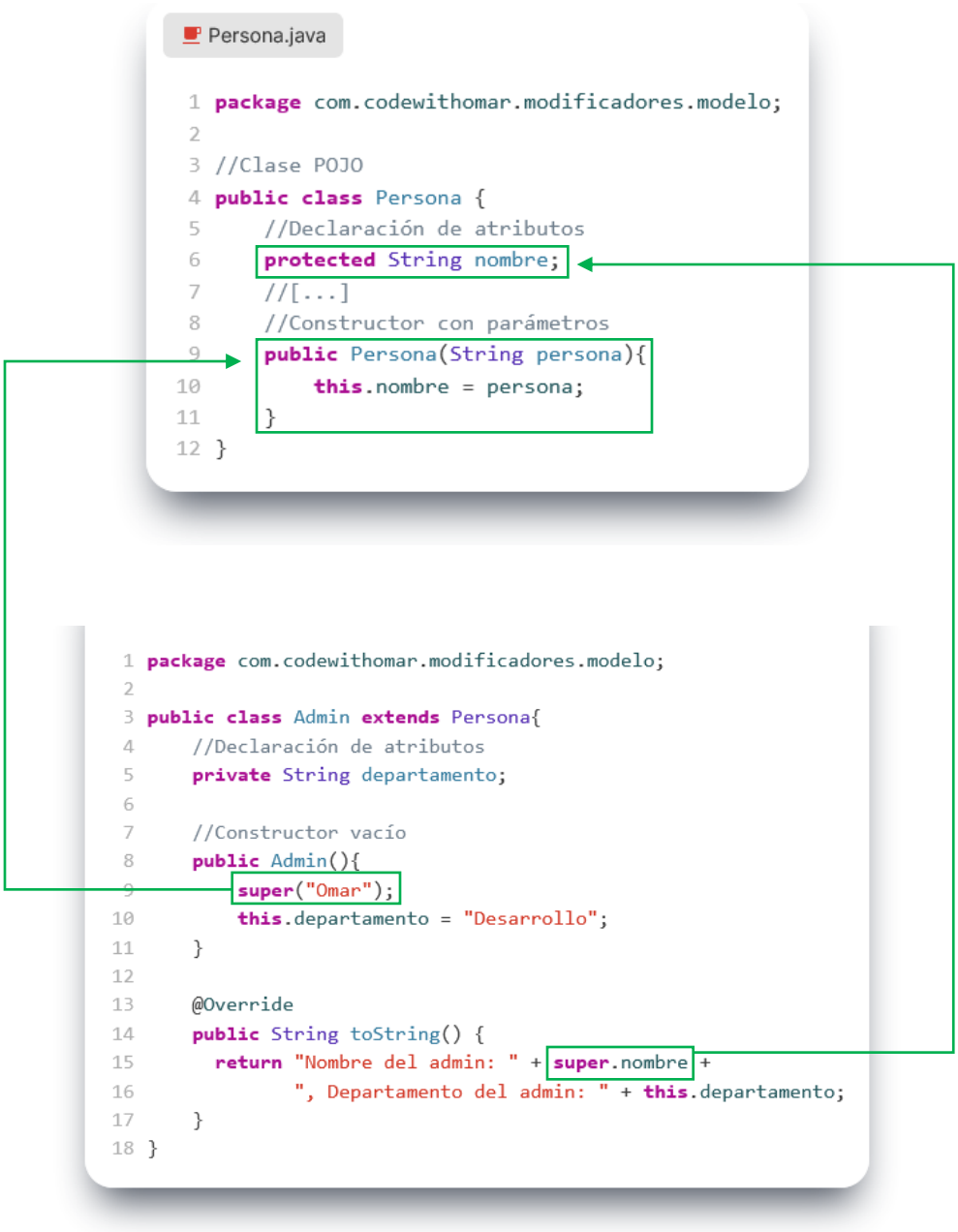
Persona.java

```java
package com.codewithomar.modificadores.modelo;

//Clase POJO
public class Persona {
    //[...]
    public Persona(){}

    public setNombre(String nombre){
      this.nombre = nombre;
    }

    public getNombre(){
      return this.nombre;
    }
    //[...]
}
```

2.3.2. Cuando usar modificador de acceso protected

El modificador de acceso protected se deberá usar de la siguiente manera:

- Los atributos de las superclases (clases Padres) para que sus subclases (clases Hijas) puedan tener acceso a ellas directamente y asignarles valor invocando el constructor de la superclase con la palabra clave super y utilizar esos atributos en otros métodos que los necesiten.
  De lo contrario, si fuesen declarados como private (siguiendo las reglas de encapsulamiento) sus clases hijas no tendrían acceso a los atributos de la superclase.



2.3.3. Cuando usar modificador de acceso default

El modificador de acceso default se deberá usar de la siguiente manera:

- Se suele utilizar métodos con modificador de acceso default al declarar los métodos de una interfaz. Las interfaces serán explicadas en clases siguientes.

Comparador.java

```java
package com.codewithomar.modificadores.modelo;

//Interface que define los métodos para comparar Objetos
public interface Comparador<T> {
    boolean menorQue(T objeto);

    boolean menorIgualQue(T objeto);

    boolean igualQue(T objeto);

    boolean mayorQue(T objeto);

    boolean mayorIgualQue(T objeto);
}
```

2.3.4. Cuando usar modificador de acceso private

El modificador de acceso private se deberá usar de la siguiente manera:

- Para cumplir los conceptos de encapsulamiento, se deberá utilizar private para los atributos de una clase *que no estén relacionados con herencia de otras clases hijas*. Si otra clase desea acceder a los datos de estos atributos deberá hacerlo mediante el uso de los métodos getter y setter del atributo que serán public.

```java
package com.codewithomar.modificadores.modelo;

//Clase POJO
public class Cliente {
    //Declaración de atributos
    private String nombre;
    private String apellido;
    private String cedula;
    private Pedidos pedidos;
    //[...]
}
```

- Para declarar subrutinas. Puede haber casos en los que necesitemos métodos que ejecuten tareas simples que se repiten en varios métodos dentro de la misma clase. Por ejemplo, un método que lea un String, lo convierta en entero, lo multiplique por dos y lo vuelva a convertir en String. Como estos métodos solo se necesitan dentro de la clase y no serán utilizados por otras clases **directamente**, se pueden declarar como private.

Banco.java

```java
package com.codewithomar.modificadores.modelo;

//Clase POJO
public class Banco {
    //[..]
    //Método auxiliar
    private String subrutina(String numero){
      int numero = Integer.parseInt(numero);
      numero = numero * 2;
      String resultado = String.getValueOf(numero);
      return resultado;
    }
    //[...]
}
```