Índice de contenido

1. Paquetes en Java

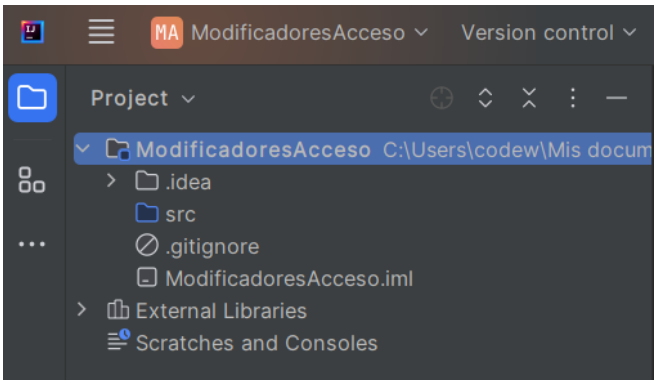
En Java, un paquete es un mecanismo utilizado para organizar y agrupar clases relacionadas. Los paquetes proporcionan un espacio de nombres para las clases, lo que ayuda a evitar conflictos de nombres y facilita la organización del código.

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

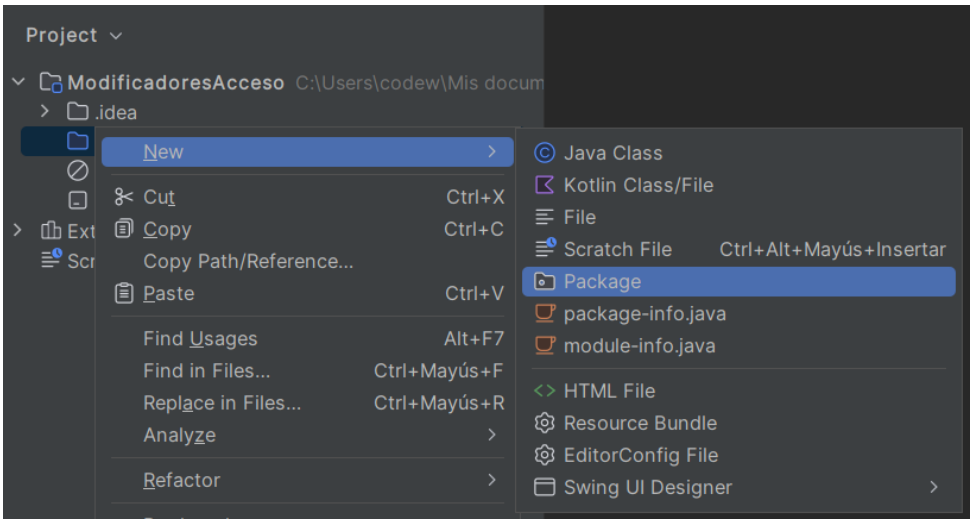
En programas mas complejos y empresariales pueden existir cientos de clases en un mismo proyecto por lo que seria demasiado complicado tener todas las clases en “src” de nuestro proyecto.

1.1. Ejemplo: Creación de un paquete desde IntelliJ

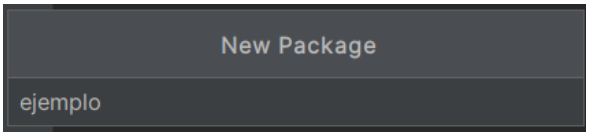
Creamos un nuevo proyecto en IntelliJ llamado “ModificadoresAcceso”.



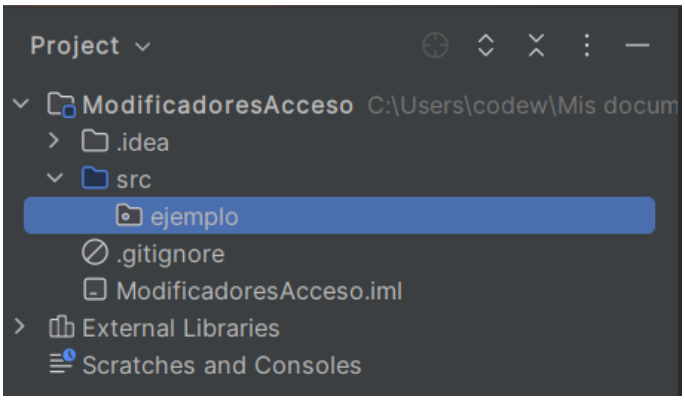
Damos click derecho sobre la carpeta “src” y seleccionamos New -> Package



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.

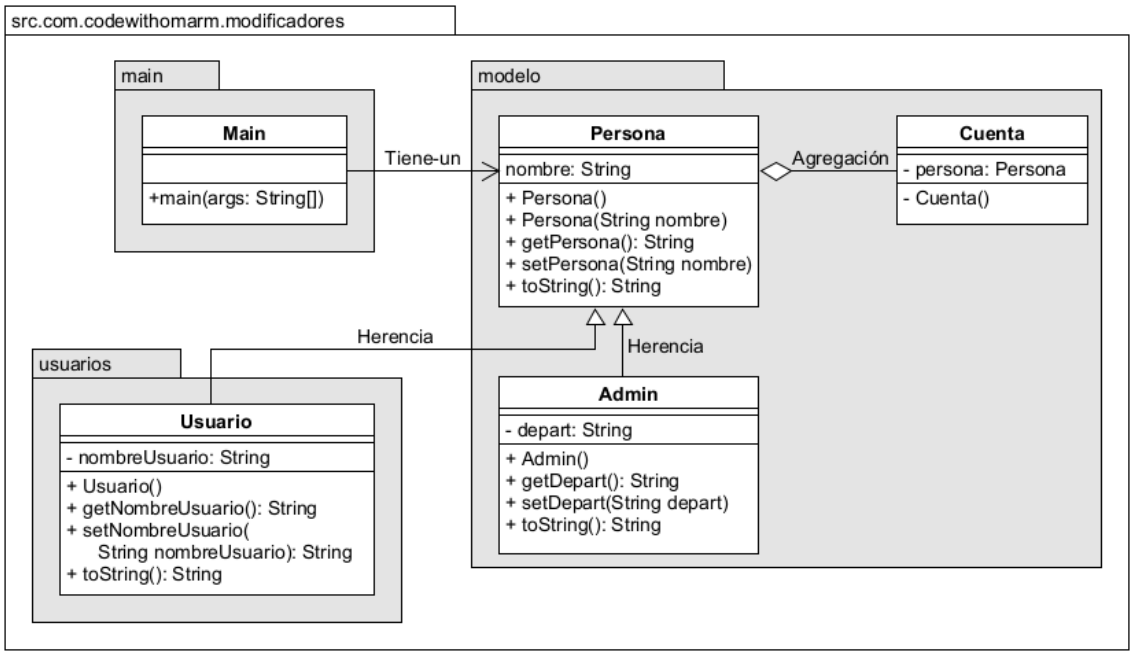
2.2. Tipos de modificadores de acceso

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:

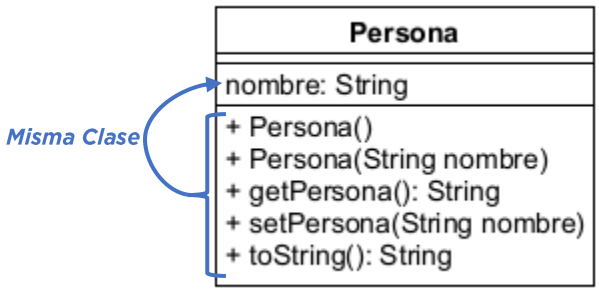


2.2.1. Explicación Niveles de Acceso

Nivel de acceso: Misma clase

Se refiere a todos los elementos que quieran acceder al elemento desde la misma clase donde es declarada.

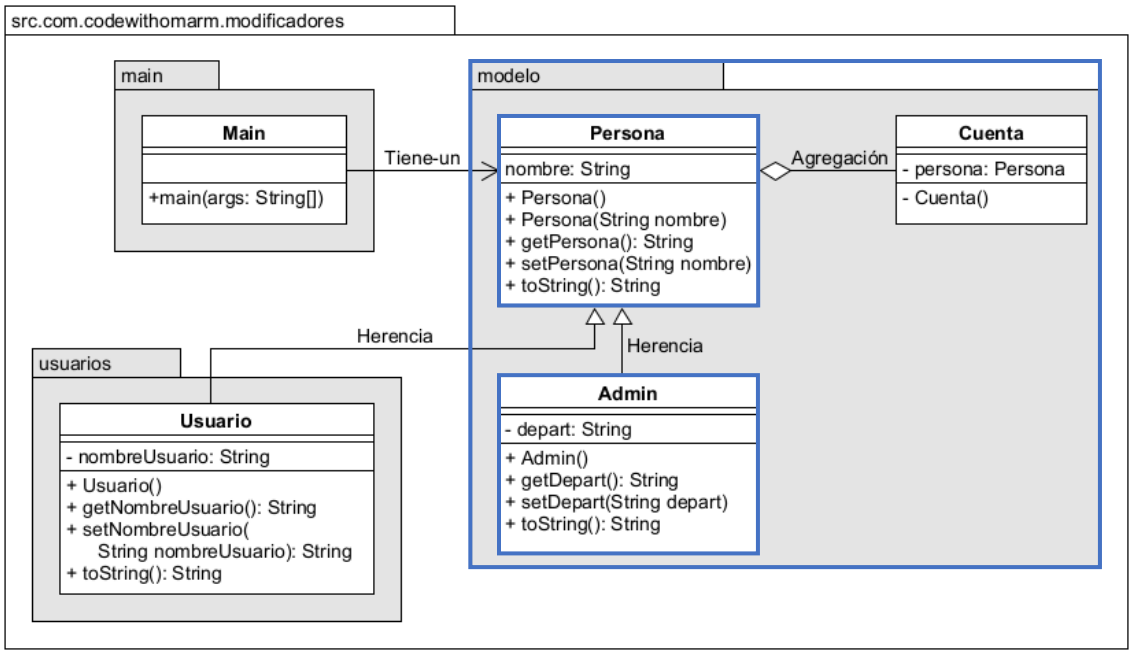
Por ejemplo: La clase persona tiene declarado un atributo llamado *nombre* de tipo String. Todos los métodos que están declarados dentro de la clase Persona se encuentran en el nivel de acceso “Misma Clase”.



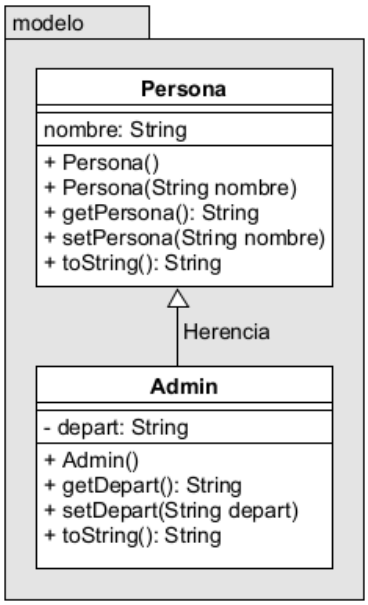
Nivel de acceso: Subclase/Herencia en el mismo paquete

Este nivel de acceso hace referencia en el caso de que una superclase (padre) y subclase (hija) 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:

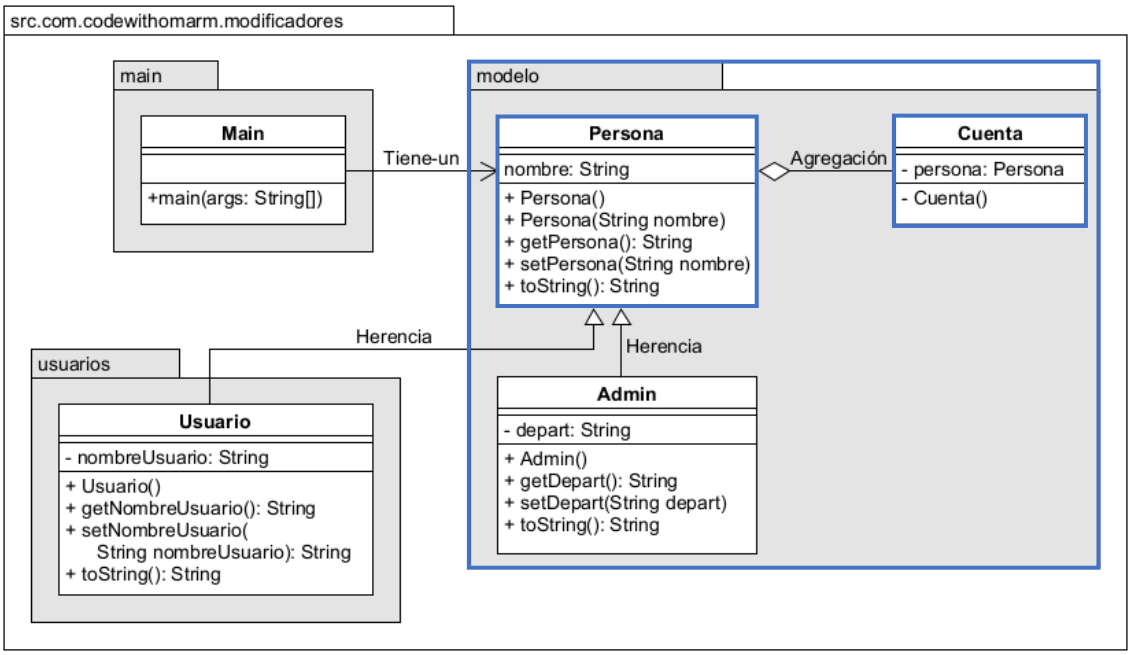


Nivel de acceso: Otra clase en el mismo paquete

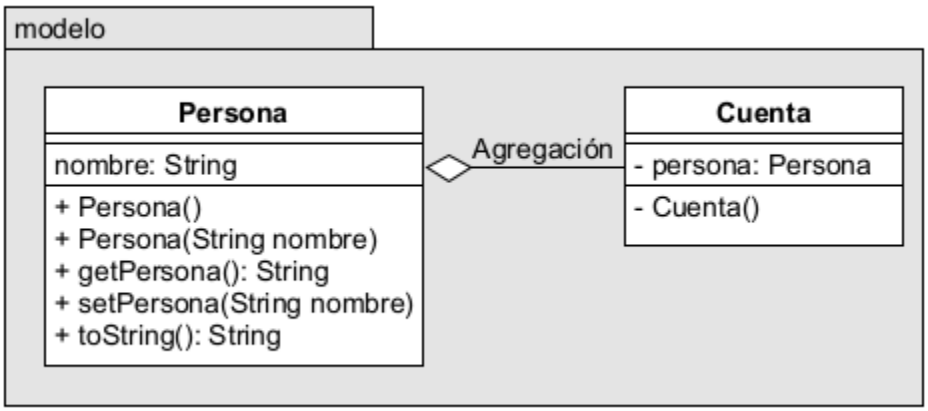
Hace referencia cuando una clase y otra clase (**que no existe herencia entre ellas**) se encuentran dentro de exactamente el mismo paquete.

El paquete "modelo" contiene a las clases "Persona" y "Cuenta". Entre Persona y Cuenta no existe una relación de herencia.

Dentro de la clase "Cuenta" se declara un objeto *private* de tipo "Persona" llamado "persona". Debido a este atributo existe una relación de agregación entre "Persona" y "Cuenta".



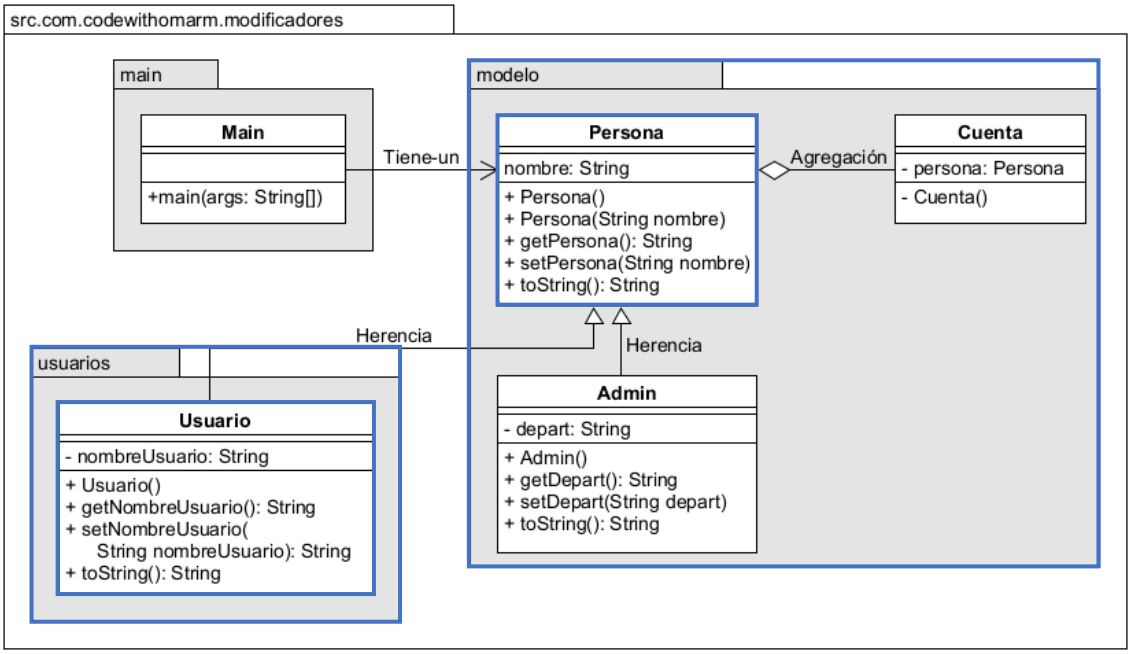
Las relaciones de agregación se diagraman en lenguaje UML utilizando la línea continua y el rombo vacío del lado de la clase a la que pertenece el objeto. Tomando solo estas dos clases podríamos verlo en UML asi:



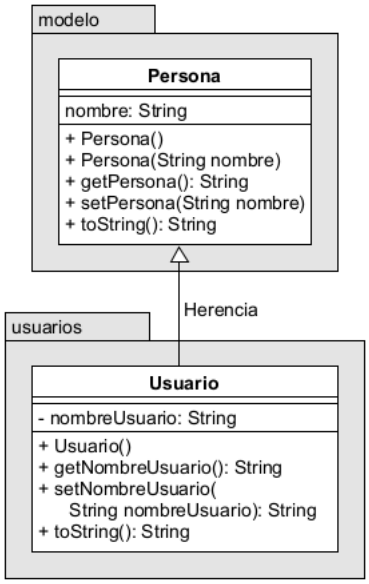
Nivel de acceso: Subclase/Herencia en otro paquete

Hace referencia cuando 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 subclase "usuario" se encuentra dentro del paquete "usuarios", por lo que ambas se encuentran dentro de distintos paquetes.



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 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.

Dentro de la clase "Main" se encuentra el método "main" que utilizará un objeto del tipo Persona, por lo que tienen una relación de composición.

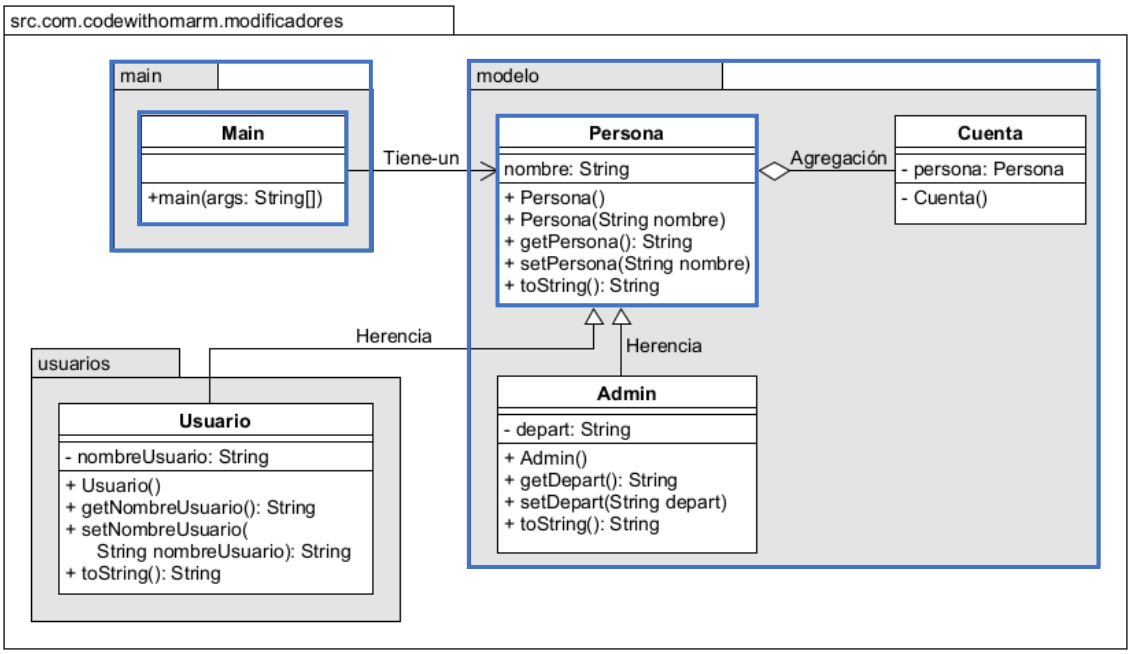
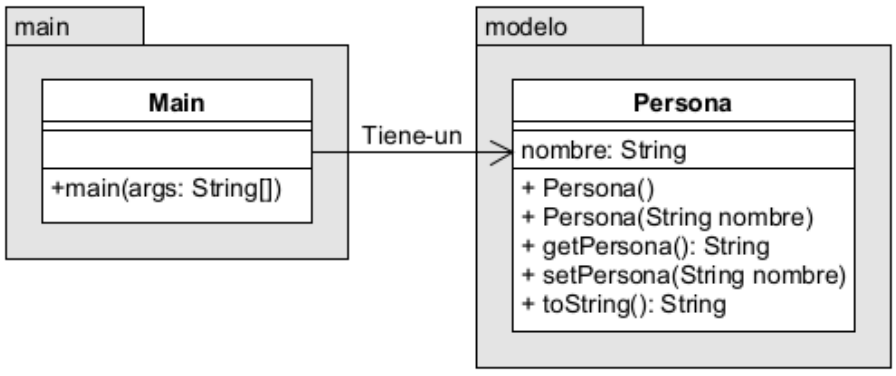


Diagrama UML solo con estos dos paquetes y clases:



2.2.2. Modificador de acceso Public

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

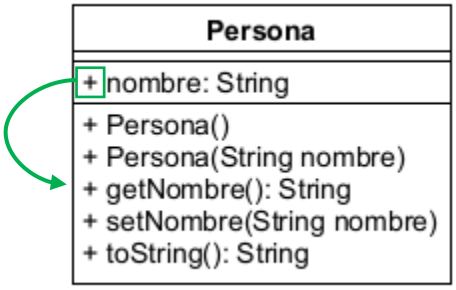
		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Public	+	✓	✓	✓	✓	✓

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

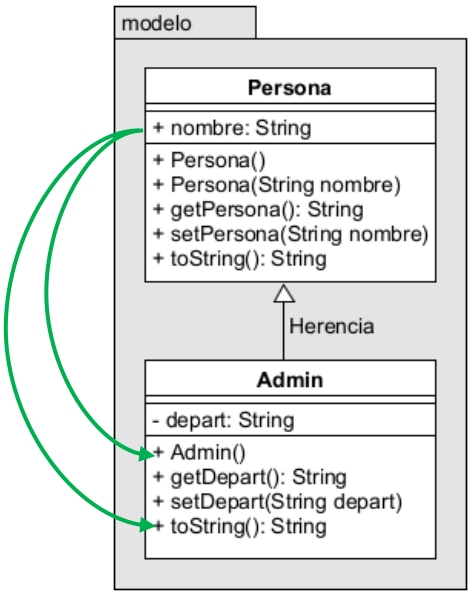


Tenemos declarado al atributo nombre con acceso "public", por lo que puede ser llamado desde otro elemento de la misma clase, en este caso los métodos constructores, getters, setters y toString

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

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

    //Constructor sin parámetros
    public Persona(){
        this.nombre = "Omar";
    }
```

```
12
13      //Constructor con parámetros
14      public Persona(String nombre){
15          this.nombre = nombre;
16      }
17
18      //Getter atributo persona
19      public String getNombre() {
20          return nombre;
21      }
22
23      //Setter atributo nombre
24      public void setNombre(String nombre) {
25          this.nombre = nombre;
26      }
27
28      //ToString objeto persona
29      @Override
30      public String toString() {
31          return "Persona{" +
32                  "nombre=" + nombre + '\'' +
33                  '}';
34      }
35  }
```

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



A través de la herencia la clase Admin tiene acceso a todos los métodos y atributos en la clase Persona.

La clase Admin llama directamente el atributo "nombre" en los método constructor "Admin()" y método "toString()".

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

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

    //[...]
}
```

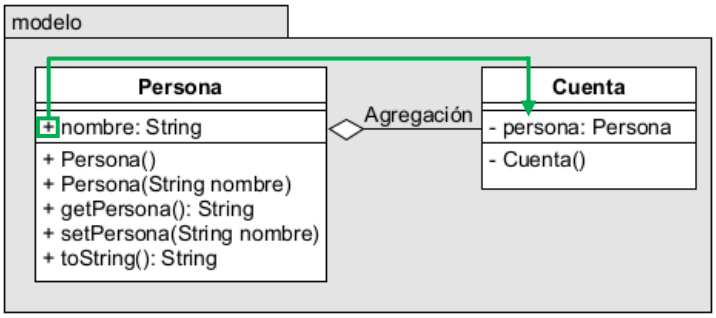
La herencia entre dos clases se declara utilizando la palabra clave *extends*.

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 y 25, 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.

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

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

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

    public String getDepartamento() {
        return departamento;
    }

    public void setDepartamento(String departamento) {
        this.departamento = departamento;
    }

    @Override
    public String toString() {
        return "Admin{" +
                "departamento='" + departamento + '\'' +
                ", nombre='" + super.nombre + '\'' +
                '}';
    }
}
```

✓ Acceso al elemento desde clase en el mismo paquete




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

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

    //[...]
}
```

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

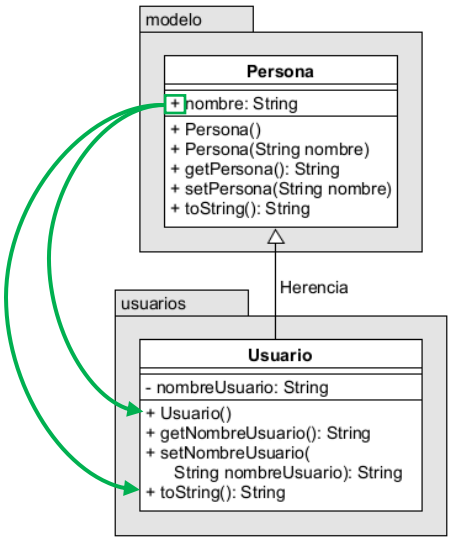
En la clase Cuenta se declara un objeto de la clase Persona. Al crearse un objeto dentro de otra clase, **esta clase podrá llamar a los elementos declarados en la clase del objeto**.

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.

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

public class Cuenta {
    //Declaración de atributos
    private Persona persona;

    //Constructor vacío
    public Cuenta(){
        persona = new Persona();
        persona.nombre = "Omar M";
    }
}
```

✓ Acceso al elemento desde su subclase en otro paquete




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

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

    //[...]
}
```

La clase Usuario se encuentra dentro del paquete "usuarios" y extiende de la clase Persona que se encuentra dentro del paquete "modelo".

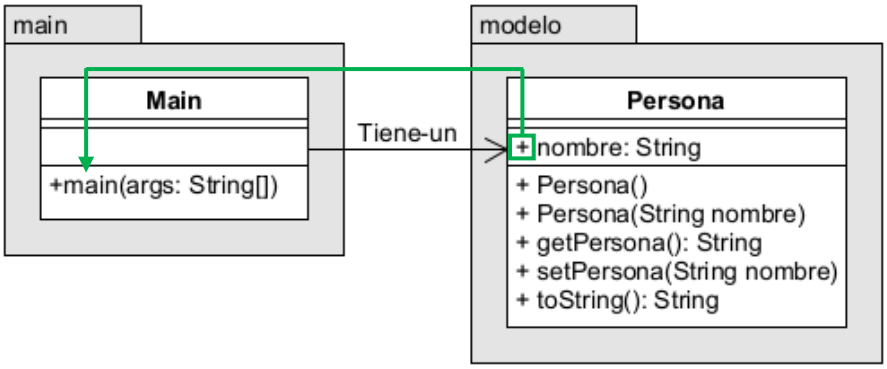
Como Usuario es hijo de Persona, puede llamar a su atributo "nombre" utilizando la palabra clave "super". Como el atributo nombre de Persona tiene modificador de acceso "public", la clase hija desde otro paquete puede llamarlo directamente.

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.

```java
package com.codewithomar.modificadores.publico.usuarios;

import com.codewithomar.modificadores.modelo.Persona;

public class Usuario extends Persona {
    private String nombreUsuario;

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

```

```
13      public String getNombreUsuario() {
14          return nombreUsuario;
15      }
16
17      public void setNombreUsuario(String nombreUsuario) {
18          this.nombreUsuario = nombreUsuario;
19      }
20
21      @Override
22      public String toString() {
23          return "Usuario{" +
24                  "nombreUsuario='" + nombreUsuario + '\'' +
25                  ", nombre='" + super.nombre + '\'' +
26                  '}';
27      }
28  }
```

- ✓ Acceso al elemento desde otra clase en otro paquete

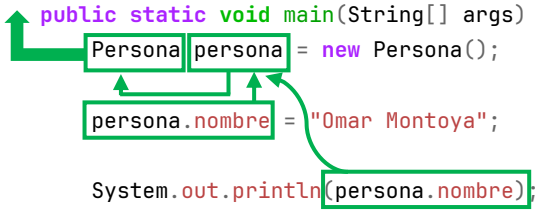



```
1   package com.codewithomar.modificadores.publico.modelo;
2
3   //Clase POJO
4   public class Persona {
5       //Declaración de atributos
6       public String nombre; //Modificador de acceso public
7
8       //[...]
9   }
```

La clase Main se encuentra dentro del paquete “main”. El paquete “main” no forma parte del paquete “modelo” donde se encuentra “Persona”. La clase main instancia un objeto de Persona para utilizar su atributo nombre, asignarle un valor y luego imprimirlo en consola.

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.

```java
package com.codewithomar.modificadores.publico.main;

import com.codewithomar.modificadores.modelo.Persona;

public class Main {
    public static void main(String[] args) {
        Persona persona = new Persona();

        persona.nombre = "Omar Montoya";

        System.out.println(persona.nombre);
    }
}
```



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.

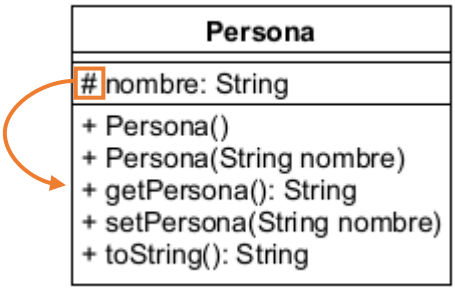
		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Protected	#	✓	✓	✓	✓	X

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 con respecto a todas las clases que no se encuentren dentro de su mismo paquete y a las subclases en otros paquetes.

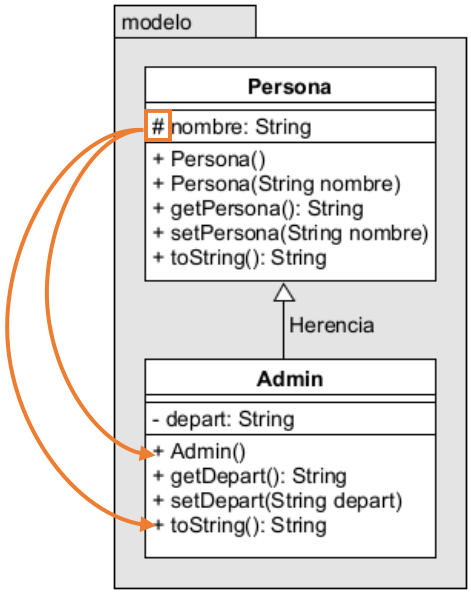
Si declaramos nuestro atributo como **protected SI** podemos tener:

- ✓ Acceso al elemento desde la misma clase.



Tenemos declarado al atributo nombre con acceso "protected", por lo que puede ser llamado desde otro elemento de la misma clase, en este caso los métodos constructores, getters, setters y toString.

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

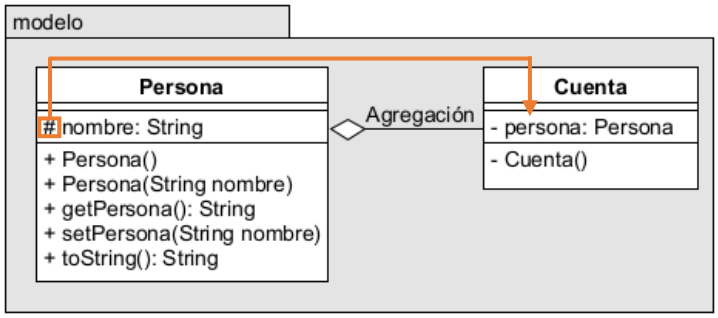


A través de la herencia la clase Admin tiene acceso a todos los métodos y atributos en la clase Persona.

El atributo nombre esta declarado como "protected" por lo que la clase dentro del mismo paquete y las clases hijas pueden acceder a ellas. En este caso la clase "Admin" cumple ambas condiciones a la vez.

La clase Admin llama directamente el atributo "nombre" en los método constructor "Admin()" y método "toString()".

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

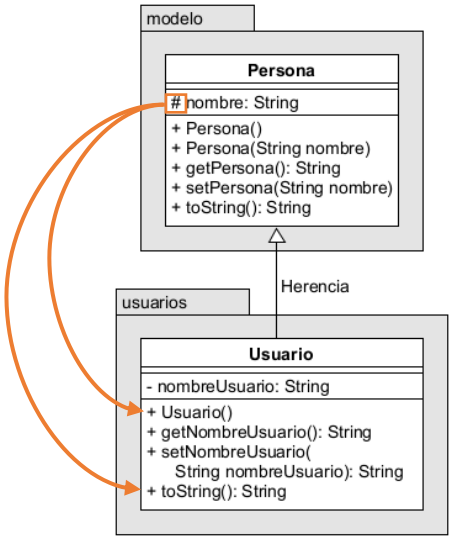


La clase Cuenta utiliza un atributo de tipo Persona.

Persona y Cuenta no tienen una relación de herencia entre ellas, por lo que no cumplen con esta condición para poder acceder a los elementos protected de Persona.

Persona y Cuenta si se encuentran dentro del mismo paquete "modelo", por lo que si cumple con la segunda condición para poder acceder directamente a los elementos protected de la clase Persona.

✓ Acceso al elemento desde una subclase en otro paquete.

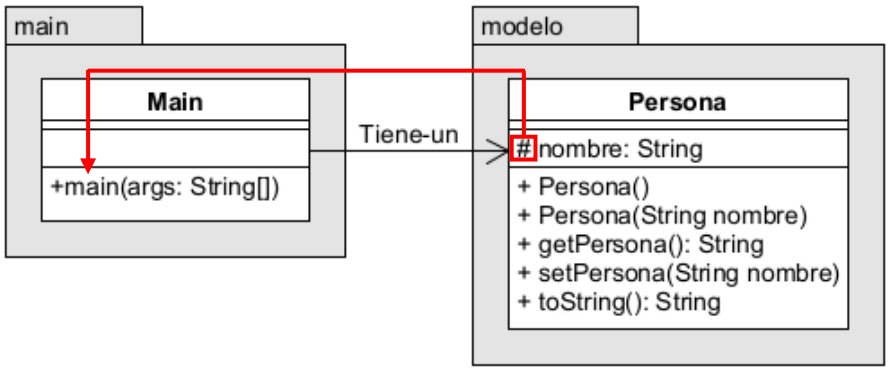


La clase Cuenta extiende/hereda de la clase Usuario.

Persona y Cuenta no se encuentran dentro del mismo paquete "modelo", Sin embargo, como se está usando protected y tienen una relación de herencia entre las dos clases se puede acceder a sus elementos.

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

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




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

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

    //[...]
}
```

La clase Persona se encuentra dentro del paquete "modelo". La clase Main se encuentra dentro del paquete "main". Como Persona y Main se encuentran en clases diferentes no cumplen con la condición de estar en el mismo paquete para utilizar los elementos protected directamente.

Entre la clase Persona y la clase Main no existe ninguna relación de herencia entre las clases. Por esta razón, tampoco cumple con la segunda condición para utilizar los elementos protected directamente.

Como no cumple con ninguna de estas condiciones, el método main no puede llamar directamente al atributo nombre del objeto persona directamente (Línea 9 y 11) y se ejecuta un error de compilación,

```java
package com.codewithomar.modificadores.protegido.main;

import com.codewithomar.modificadores.modelo.Persona;

public class Main {
    public static void main(String[] args) {
        Persona persona = new Persona();

        persona.nombre = "Omar Montoya";

        System.out.println(persona.nombre);
    }
}
```

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.

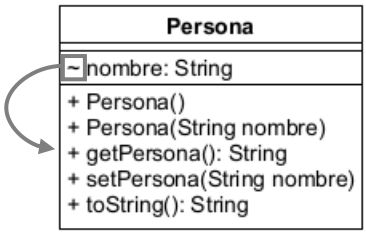
		Nivel de acceso				
		Misma Clase	Subclase/Herencia en el mismo paquete	Otra clase en el mismo paquete	Subclase/Herencia en otro paquete	Otra clase en otro paquete
Default	~	✓	✓	✓	✗	✗

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 del mismo paquete de la clase donde el elemento se encuentra declarado.

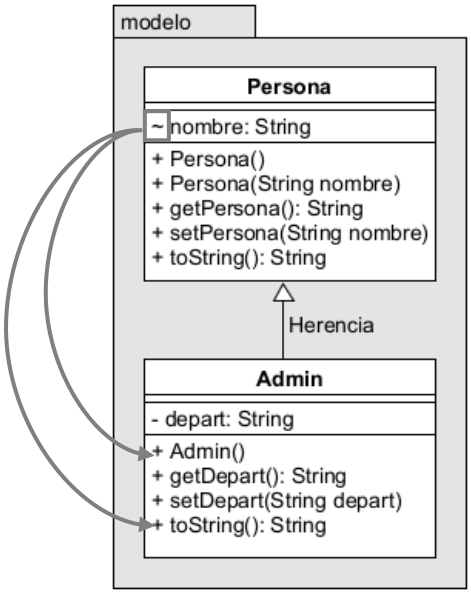
Por este motivo se le conoce como el modificador "package-private"

Si declaramos nuestro atributo como default podemos tener:

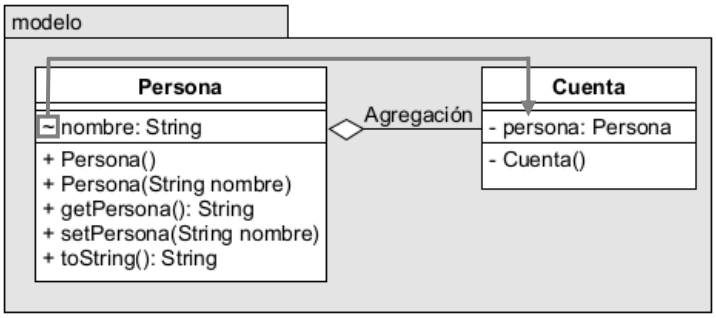
- ✓ Acceso al elemento desde la misma clase.



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

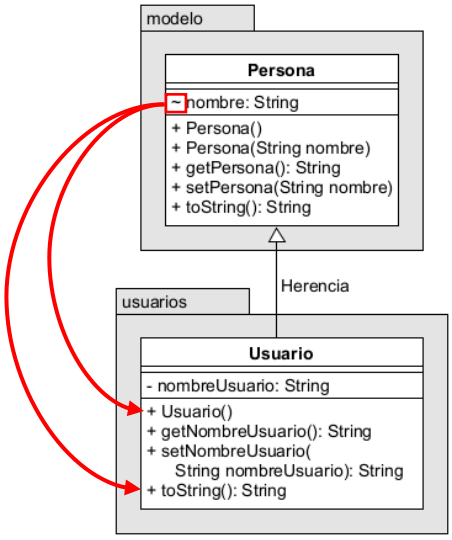


✓ Acceso al elemento desde otra clase en el mismo paquete.



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

✗ Acceso al elemento desde una subclase en otro paquete.



La clase Cuenta extiende/hereda de la clase Usuario.

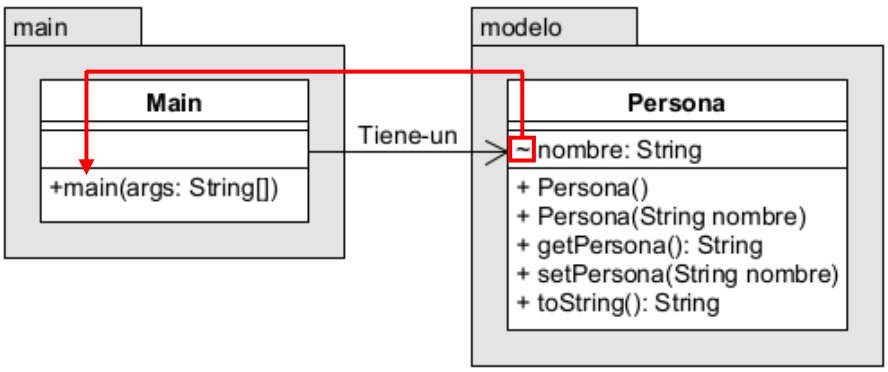
Persona y Cuenta si tienen una relación de herencia entre ellas, por lo que si cumplen con la primera condición para poder acceder a los elementos de Persona.

Sin embargo, Persona y **Cuenta no se encuentran dentro del mismo paquete**, por lo que no cumple con la segunda condición para poder acceder directamente a los elementos default de la clase Persona.

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

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

    //[...]
}
```

```java
package com.codewithomar.modificadores.defecto.usuarios;

import com.codewithomar.modificadores.defecto.modelo.Persona;

public class Usuario extends Persona {
    private String nombreUsuario;

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

    public String getNombreUsuario() {
        return nombreUsuario;
    }

    public void setNombreUsuario(String nombreUsuario) {
        this.nombreUsuario = nombreUsuario;
    }

    @Override
    public String toString() {
        return "Usuario{" +
                "nombreUsuario='" + nombreUsuario + '\'' +
                ", nombre='" + super.nombre + '\'' +
                '}';
    }
}
```

- Acceso al elemento desde otra clase en otro paquete.



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

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

    //[...]
}
```

La clase Main crea un objeto de la clase Persona.

El método main trata de llamar directamente al atributo "nombre" del objeto "persona" para asignarle un valor, sin embargo como el atributo tiene modificador de acceso default esto no es permitido.

No se le permite el acceso debido a que la clase Main y Persona **se encuentran en paquetes distintos**.

```java
package com.codewithomar.modificadores.defecto.main;

import com.codewithomar.modificadores.defecto.modelo.Persona;

public class Main {
    public static void main(String[] args) {
        Persona persona = new Persona();

        persona.nombre = "Omar Montoya";

        System.out.println(persona.nombre);
    }
}
```

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

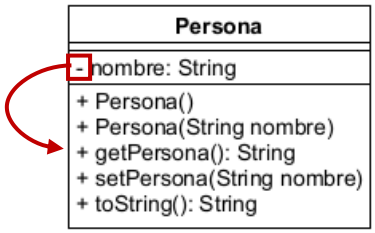
El modificador de acceso private es el modificador más restrictivo de todos. Solo se puede acceder a los elementos desde su propia clase y de ningún otro lugar.

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 **SI** podremos:

- ✓ Acceder a los elementos desde su misma clase



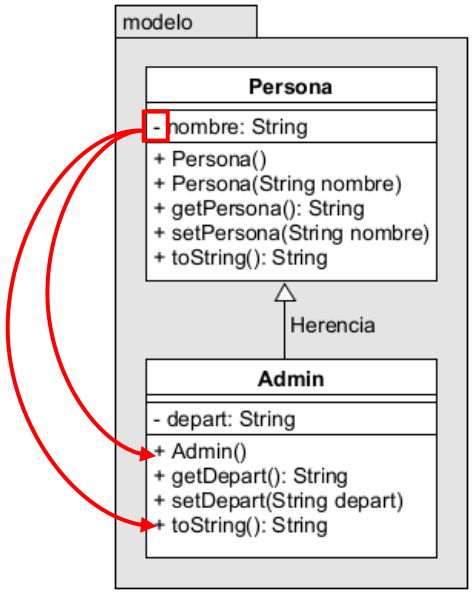
El acceso a los elementos privados de una clase se limita únicamente a los elementos de esa misma clase.

El modificador de acceso private funciona única y exclusivamente para este tipo de comportamiento.

Su finalidad es proteger los datos de la clase de cualquier clase externa que desee acceder directamente a sus elementos.

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

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

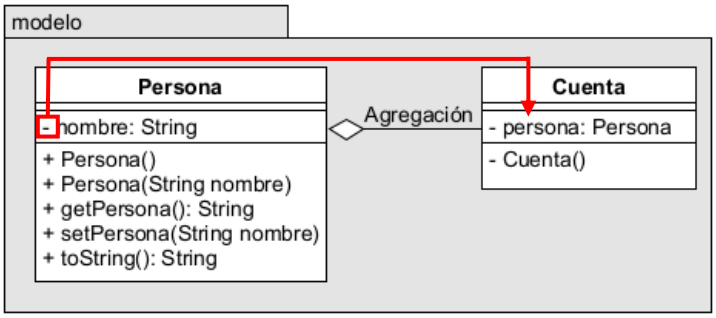



```java
package com.codewithomar.modificadores.privado.modelo;

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

    //[...]
}
```

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

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

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

    public String getDepartamento() {
        return departamento;
    }
```

```
16
17      public void setDepartamento(String departamento) {
18          this.departamento = departamento;
19      }
20
21      @Override
22      public String toString() {
23          return "Admin{" +
24                  "departamento='" + departamento + '\'' +
25                  ", nombre='" + super.nombre + '\'' +
26                  '}';
27      }
28  }
```

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

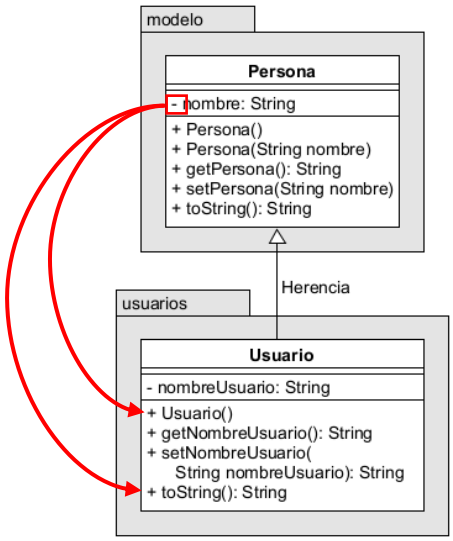



```
1  package com.codewithomar.modificadores.privado.modelo;
2
3  //Clase POJO
4  public class Persona {
5      //Declaración de atributos
6      private String nombre; //Modificador de acceso private
7
8      //[...]
9  }
```

```
1  package com.codewithomar.modificadores.privado.modelo;
2
3  public class Cuenta {
4      //Declaración de atributos
5      private Persona persona;
6
7      //Constructor vacío
8      public Cuenta(){
9          persona = new Persona();
10         persona.nombre = "Omar M";
11     }
```

```
12 }
```

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

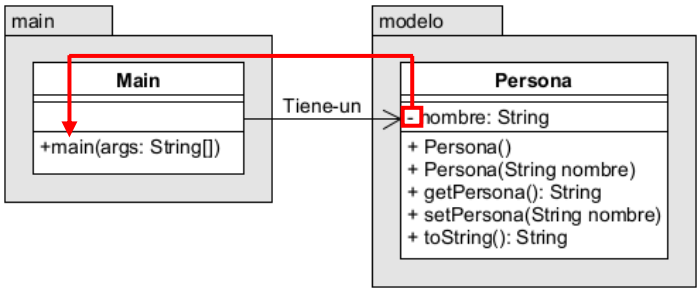



```java
package com.codewithomar.modificadores.privado.modelo;

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

    //[...]
}
```

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

import com.codewithomar.modificadores.privado.modelo.Persona;

public class Usuario extends Persona {
    private String nombreUsuario;

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

    public String getNombreUsuario() {
        return nombreUsuario;
    }

    public void setNombreUsuario(String nombreUsuario) {
```

```
18          this.nombreUsuario = nombreUsuario;
19      }
20
21      @Override
22      public String toString() {
23          return "Usuario{" +
24                  "nombreUsuario='" + nombreUsuario + '\'' +
25                  ", nombre='" + super.nombre + '\'' +
26                  '}';
27      }
28  }
```

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




```
1   package com.codewithomar.modificadores.privado.modelo;
2
3   //Clase POJO
4   public class Persona {
5       //Declaración de atributos
6       private String nombre; //Modificador de acceso private
7
8       //[...]
9   }
```

```
1   package com.codewithomar.modificadores.privado.main;
2
3   import com.codewithomar.modificadores.privado.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          System.out.println(persona.nombre);
12      }
13  }
```

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 (el dueño de Java) 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. ¿Cuándo 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 más común.**

```
1  package com.codewithomar.modificadores.privado.modelo;
2
3  public class Persona {
4      //[..]
5  }
```

- Los métodos necesarios para manipular los objetos en otras clases (Constructor, getter, setters, toString, etc) deberán ser declarados **public**. Esto garantiza el acceso a los métodos desde cualquier parte del proyecto.

```
1  package com.codewithomar.modificadores.privado.modelo;
2
3  //Clase POJO
4  public class Persona {
5      //Declaración de atributos
6      private String nombre; //Modificador de acceso private
7
8      //Constructor sin parámetros
9      public Persona(){
10         this.nombre = "Omar";
```

```
    }

    //Constructor con parámetros
    public Persona(String nombre){
        this.nombre = nombre;
    }

    //Getter atributo persona
    public String getNombre() {
        return nombre;
    }

    //Setter atributo nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    //ToString objeto persona
    @Override
    public String toString() {
        return "Persona{" +
                "nombre=" + nombre + '\'' +
                '}';
    }
}
```

2.3.2. ¿Cuándo 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 esos atributos directamente.

  De esta manera, se permitiría hacer uso de la palabra clave super para que las subclases asignen valores o utilicen los valores del atributo de la superclase.

  De lo contrario, si fuesen declarados como private (siguiendo las reglas de encapsulamiento) sus clases hijas no tendrían acceso directo a los atributos de la superclase.

  Esto es **OPCIONAL.** También se puede ser más estrictos y utilizar modificador de acceso private y acceder o modificar los valores de los atributos de la superclase utilizando constructores, getters y setters declarados public.

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

//Clase POJO
```

```java
public class Persona {
    //Declaración de atributos
    protected String nombre; //Modificador de acceso protected

    //Constructor sin parámetros
    public Persona(){
        this.nombre = "Omar";
    }

    //Constructor con parámetros
    public Persona(String nombre){
        this.nombre = nombre;
    }

    //Getter atributo persona
    public String getNombre() {
        return nombre;
    }

    //Setter atributo nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    //ToString objeto persona
    @Override
    public String toString() {
        return "Persona{" +
                "nombre=" + nombre + '\'' +
                '}';
    }
}
```

2.3.3. ¿Cuándo usar modificador de acceso default?

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

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

```java
package com.codewithomar.modificadores;

//Interface que define metodos 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. ¿Cuándo usar modificador de acceso private?

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

- Para cumplir con los principios 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.

```
package com.codewithomar.modificadores;

//Clase POJO
public class Cliente {
    //Declaración de atributos
    private String nombre;
    private String apellido;
    private String cedula;
    private String direccion;

    public Cliente() {
    }

    public Cliente(String nombre, String apellido, String cedula, String direccion) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.cedula = cedula;
        this.direccion = direccion;
    }

    public String getNombre() {
        return nombre;
    }

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

    public String getApellido() {
        return apellido;
    }

```

```
33      public void setApellido(String apellido) {
34          this.apellido = apellido;
35      }
36
37      public String getCedula() {
38          return cedula;
39      }
40
41      public void setCedula(String cedula) {
42          this.cedula = cedula;
43      }
44
45      public String getDireccion() {
46          return direccion;
47      }
48
49      public void setDireccion(String direccion) {
50          this.direccion = direccion;
51      }
52
53      @Override
54      public String toString() {
55          return "Cliente{" +
56                  "nombre='" + nombre + '\'' +
57                  ", apellido='" + apellido + '\'' +
58                  ", cedula='" + cedula + '\'' +
59                  ", direccion='" + direccion + '\'' +
60                  '}';
61      }
62  }
```

- Para declarar subrutinas. Podríamos tener 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 valide el nombre y apellido del Cliente y concatene ambas cadenas y las retorne solo si ambos valores son válidos (Línea 18 – Método `unirNombreCliente()`). Como estos métodos solo se necesitan dentro de la clase y no serán utilizados por otras clases **directamente**, se pueden declarar como private.

```
1   package com.codewithomar.modificadores;
2
3   public class Proceso {
4       //Declaración de atributos
5       private Cliente cliente;
6
7       public Proceso(){
8       }
9
10      public Proceso(Cliente cliente){
```

```
11          this.cliente = cliente;
12      }
13
14      public String mostrarNombreCliente(){
15          return unirNombreCliente();
16      }
17
18      private String unirNombreCliente(){
19          String nombreCompleto = null;
20          if (cliente.getNombre() != null && cliente.getApellido() != null) {
21              nombreCompleto = cliente.getNombre() + cliente.getApellido();
22          } else {
23              System.out.println("Nombre del cliente imcompleto");
24          }
25          return nombreCompleto;
26      }
27  }
```

Llamado a la subrutina privada