

Trabajo Practico 5

Relaciones UML 1 a 1

Materia: Programación II

Profesor: Ariel Enferrel

Tutor: Tomas Ferro

Alumna: Laura Diaco

Comisión: 3

**Caso Práctico**

**https://github.com/LauraDiaco365/TrabajoPractico5UMLrelacion1a1.git**

Desarrollar los siguientes ejercicios en Java. Cada uno deberá incluir:

● Diagrama UML

● Tipo de relación (asociación, agregación, composición, dependencia)

● Dirección (unidireccional o bidireccional)

● Implementación de las clases con atributos y relaciones definidas

Ejercicios de Relaciones 1 a 1

**1. Pasaporte - Foto – Titular**

a. Composición: Pasaporte → Foto

b. Asociación bidireccional: Pasaporte ↔ Titular

Clases y atributos:

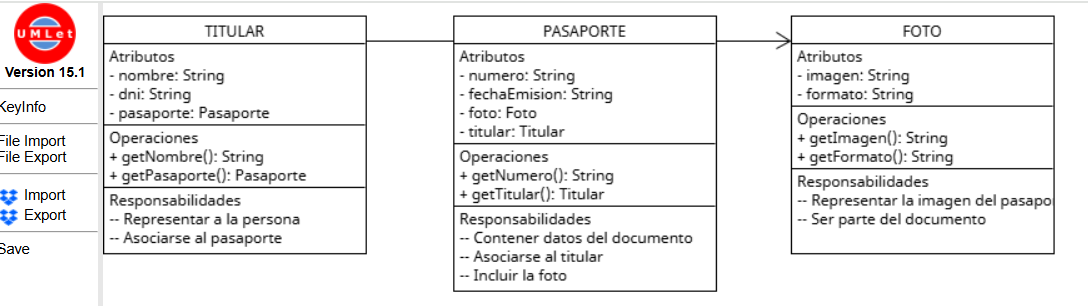
i. Pasaporte: numero, fechaEmision

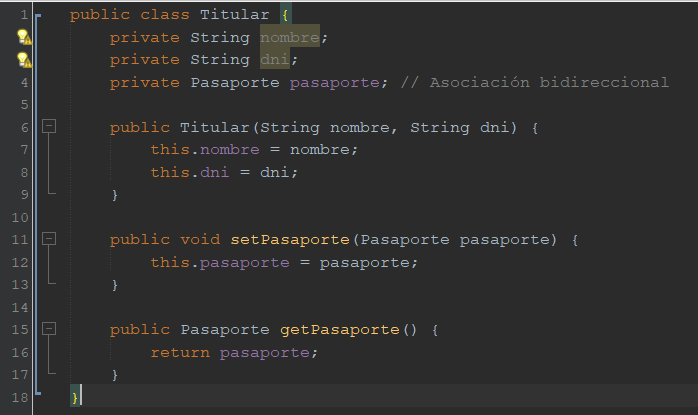
ii. Foto: imagen, formato

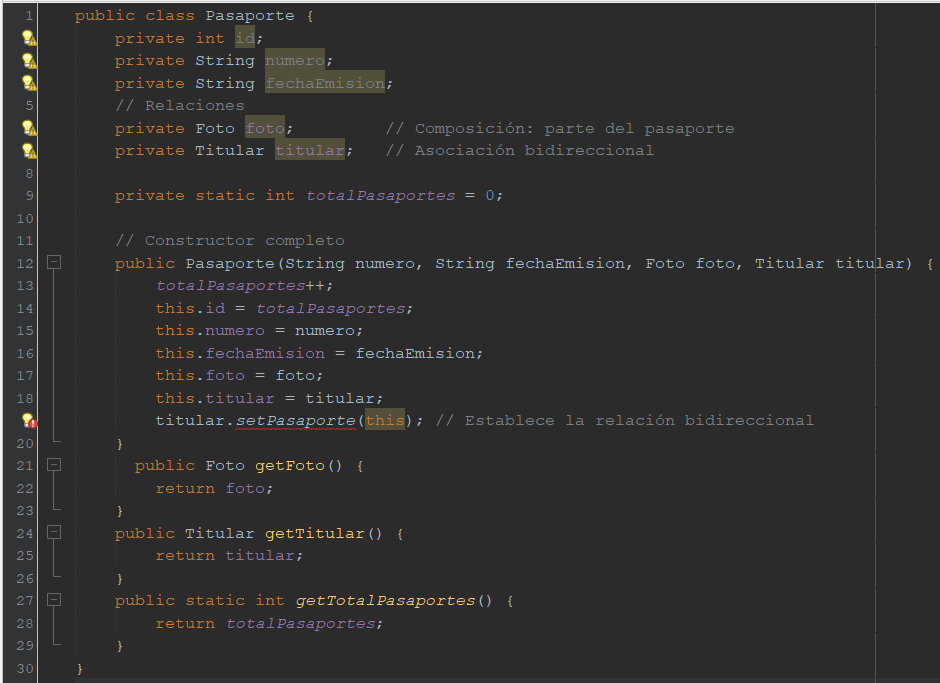
iii. Titular: nombre, dni

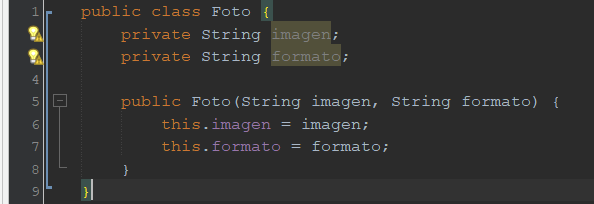
**Relaciones:**

* **Composición** (unidireccional): Pasaporte → Foto  
  → La foto es parte del pasaporte. Si el pasaporte se elimina, la foto también.
* **Asociación bidireccional**: Pasaporte ↔ Titular  
  → El titular conoce su pasaporte y el pasaporte conoce a su titular.









2. Celular - Batería – Usuario

a. Agregación: Celular → Batería

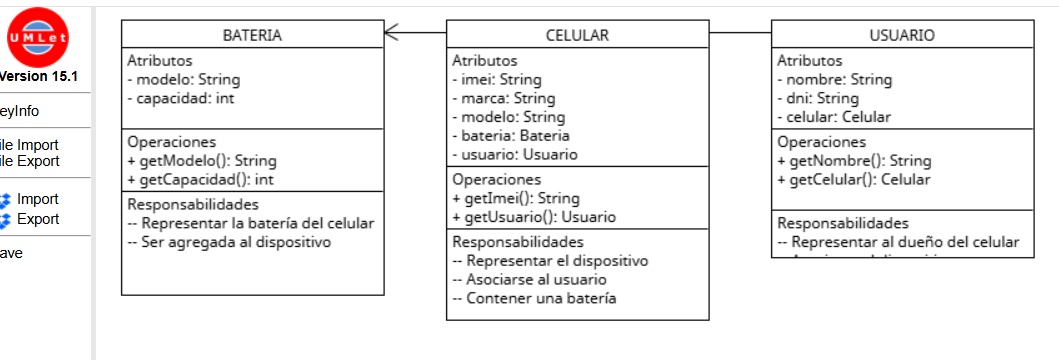
b. Asociación bidireccional: Celular ↔ Usuario

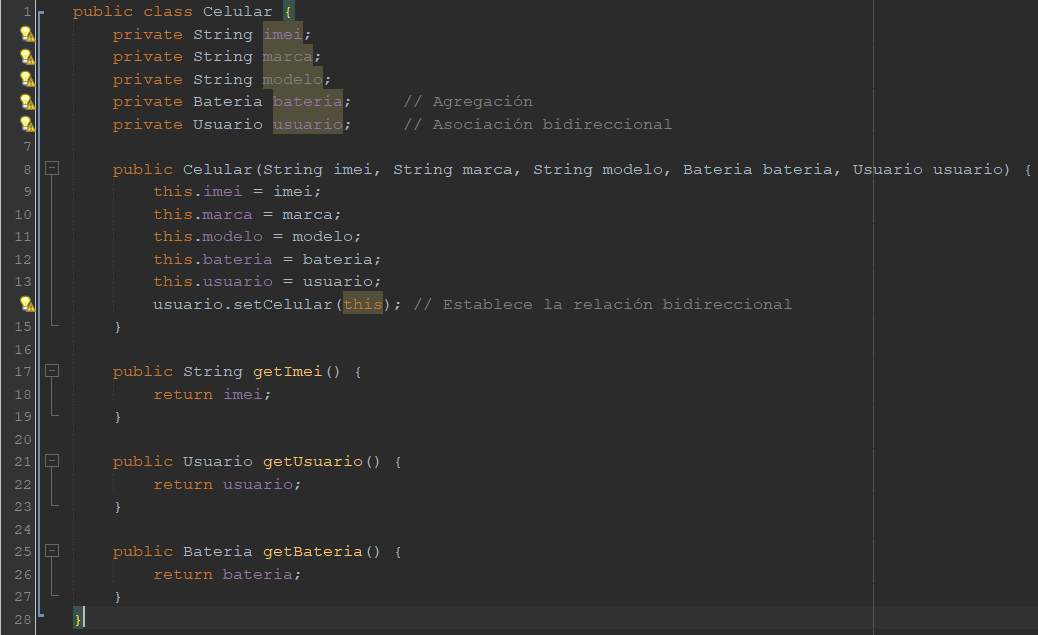
Clases y atributos:

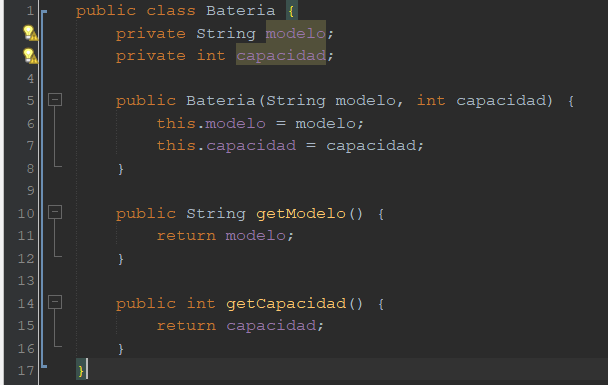
1. Celular: imei, marca, modelo
2. Batería: modelo, capacidad
3. Usuario: nombre, dni

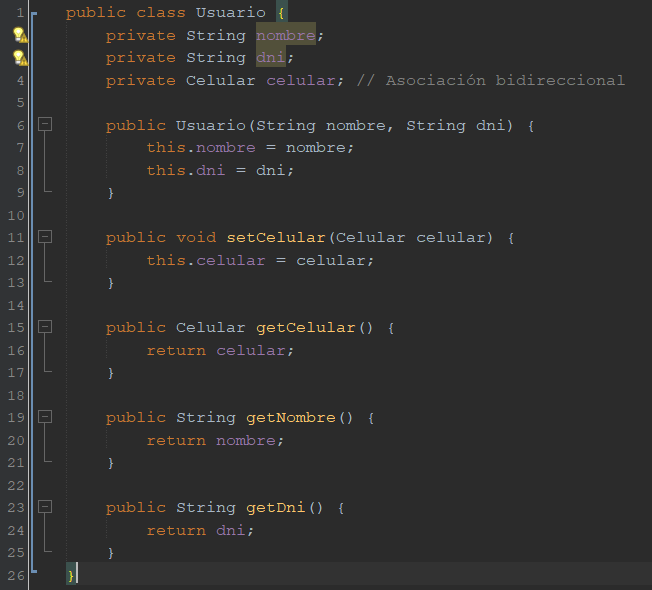
**Agregación**: Celular → Bateria (la batería puede existir por separado)

**Asociación bidireccional**: Celular ↔ Usuario (ambos se conocen mutuamente)









3. Libro - Autor – Editorial

a. Asociación unidireccional: Libro → Autor

b. Agregación: Libro → Editorial Clases y atributos:

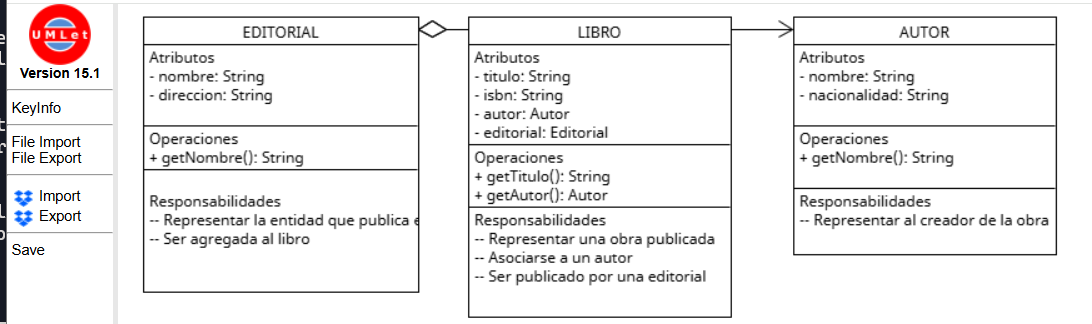
i. Libro: titulo, isbn

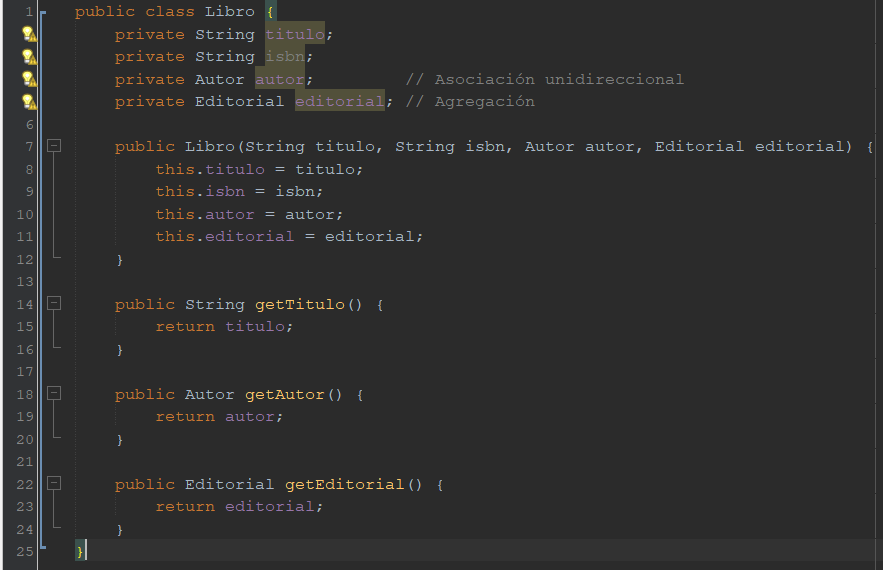
ii. Autor: nombre, nacionalidad

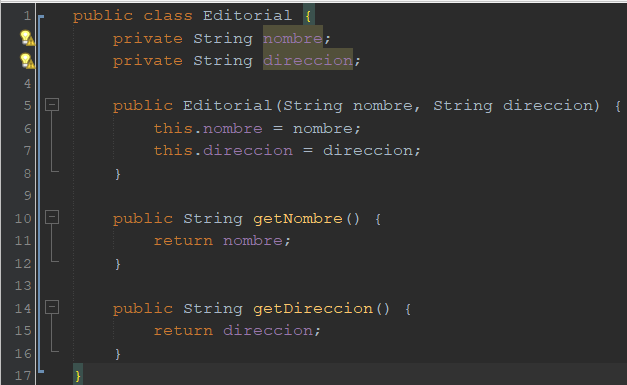
iii.Editorial: nombre, dirección

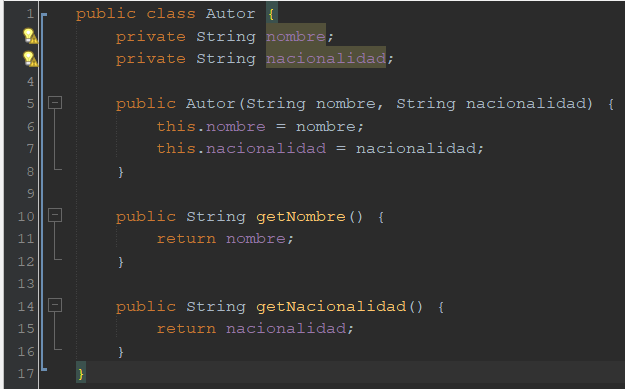
**Asociación unidireccional**: Libro → Autor (el libro conoce al autor, pero no al revés)

**Agregación**: Libro → Editorial (la editorial puede existir sin el libro)









4. TarjetaDeCrédito - Cliente – Banco

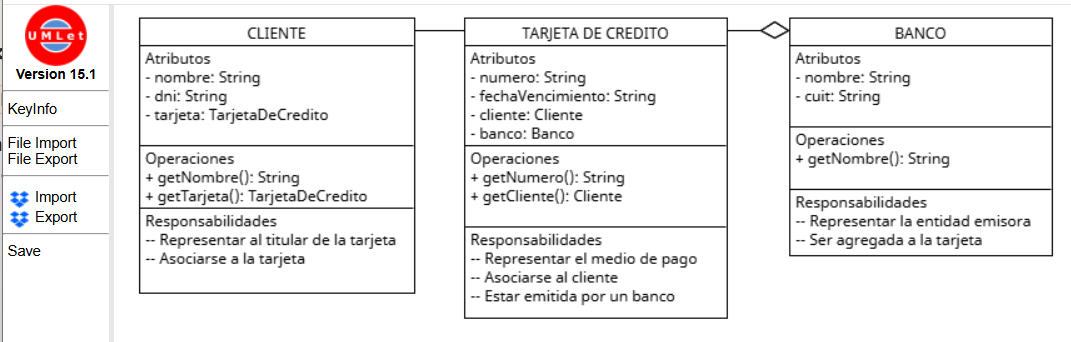
a. Asociación bidireccional: TarjetaDeCrédito ↔ Cliente

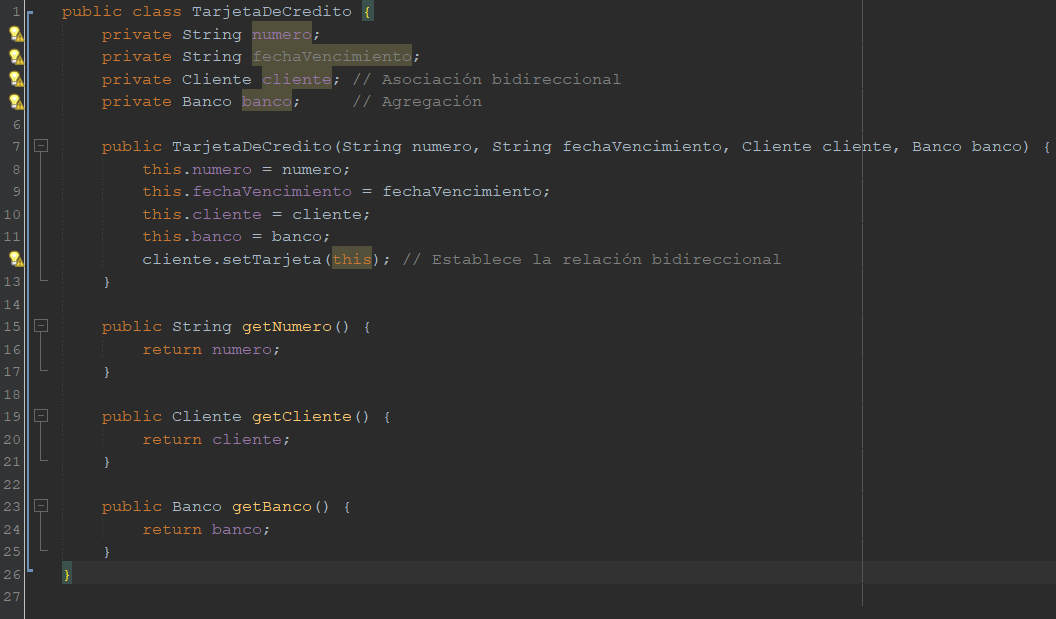
b. Agregación: TarjetaDeCrédito → Banco

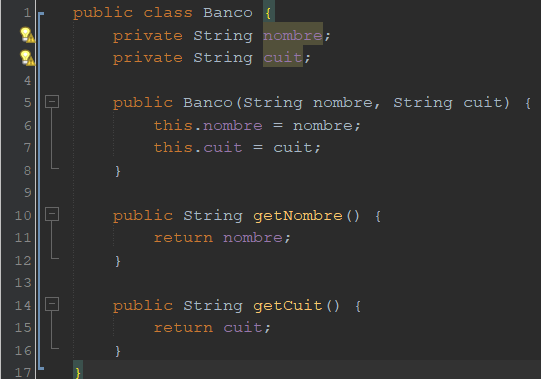
Clases y atributos:

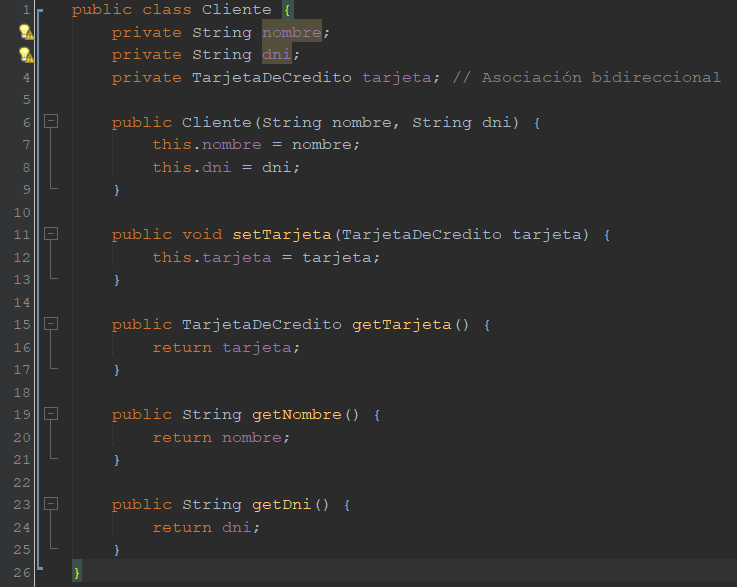
1. TarjetaDeCrédito: numero, fechaVencimiento
2. Cliente: nombre, dni
3. Banco: nombre, cuit

– **Asociación bidireccional** entre TarjetaDeCredito y Cliente, lo que significa que ambos objetos se conocen mutuamente.  
– **Agregación** entre TarjetaDeCredito y Banco, lo que implica que el banco puede existir independientemente de la tarjeta, pero la tarjeta mantiene una referencia a él.









5. Computadora - PlacaMadre - Propietario

a. Composición: Computadora → PlacaMadre

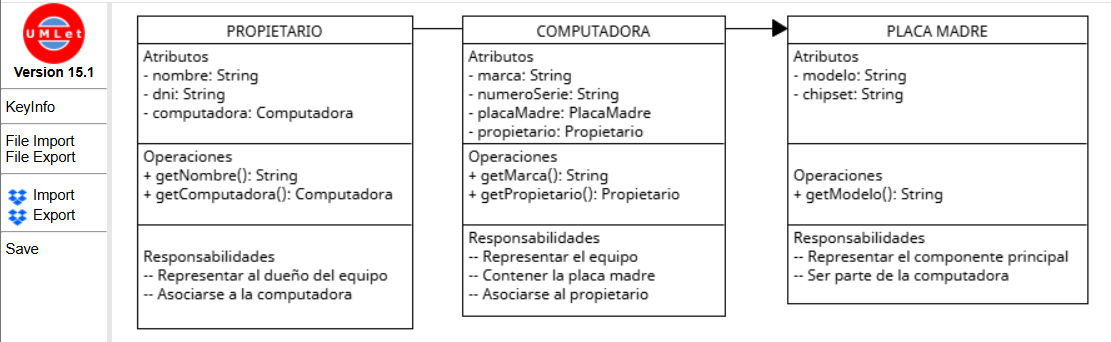
b. Asociación bidireccional: Computadora ↔ Propietario

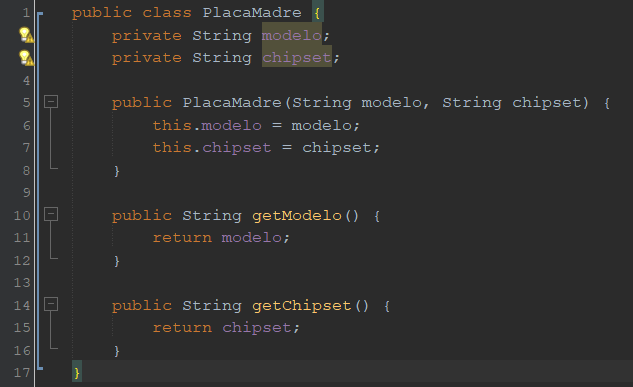
Clases y atributos:

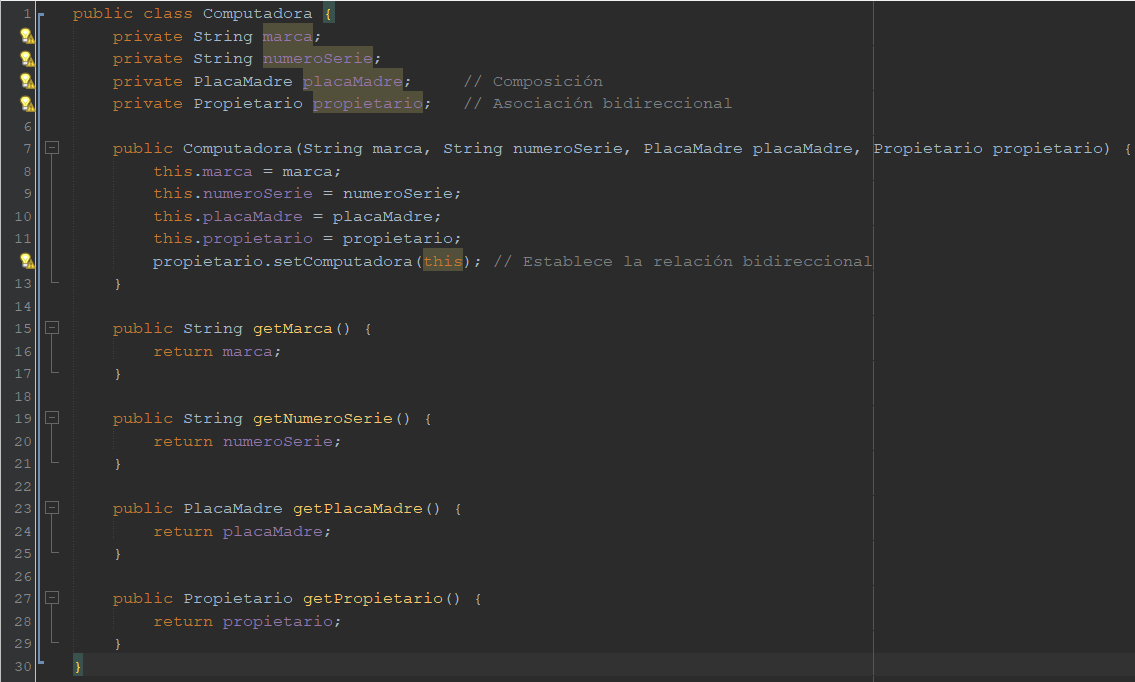
1. Computadora: marca, numeroSerie
2. PlacaMadre: modelo, chipset
3. Propietario: nombre, dni

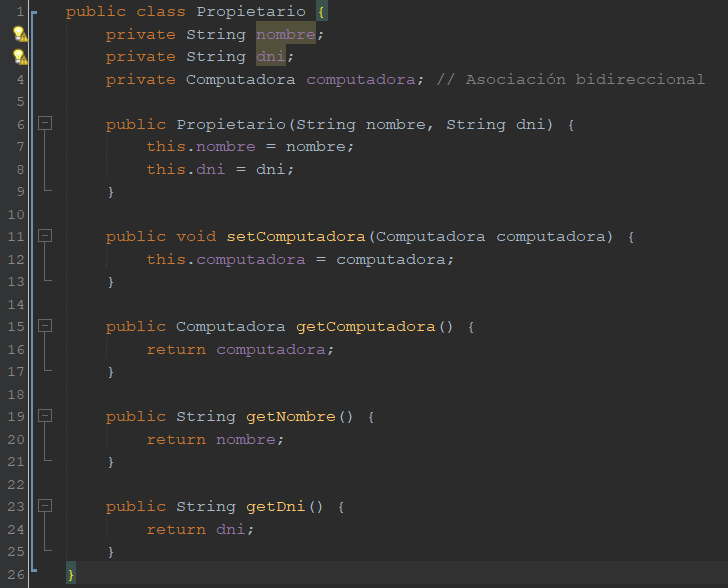
**Composición**: Computadora → PlacaMadre (la placa madre no existe sin la computadora)

**Asociación bidireccional**: Computadora ↔ Propietario (ambos se conocen mutuamente)









6. Reserva - Cliente - Mesa

a. Asociación unidireccional: Reserva → Cliente

b. Agregación: Reserva → Mesa Clases y atributos:

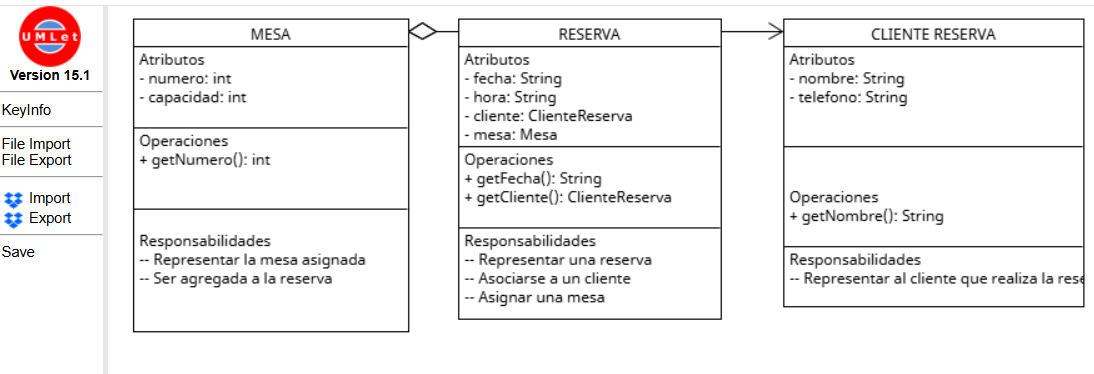
i. Reserva: fecha, hora

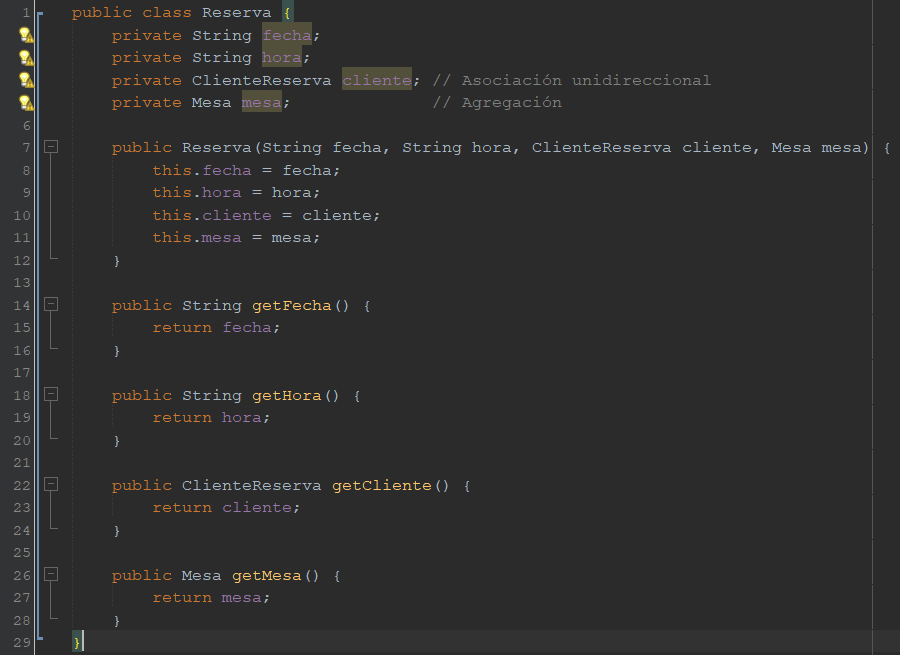
ii. Cliente: nombre, teléfono

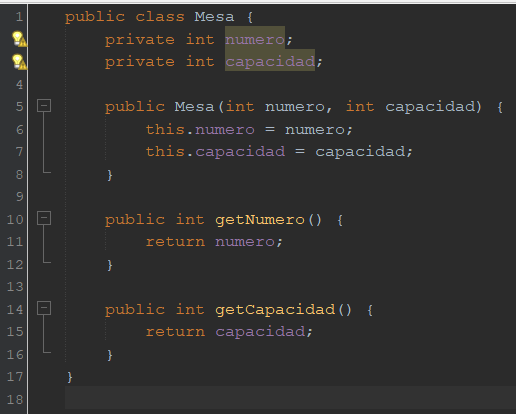
iii.Mesa: numero, capacidad

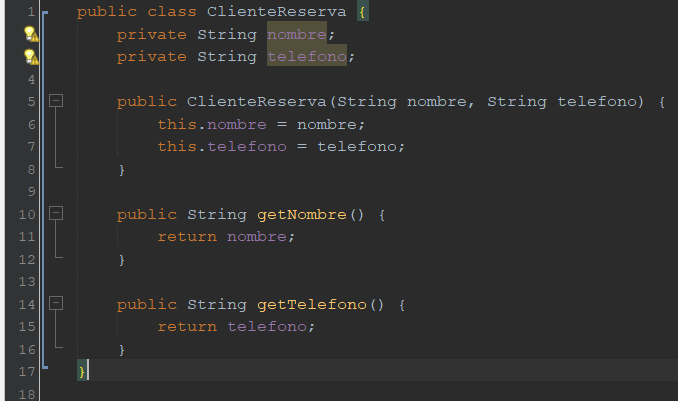
**Asociación unidireccional**: Reserva → Cliente (la reserva conoce al cliente, pero no al revés)

**Agregación**: Reserva → Mesa (la mesa puede existir sin la reserva)









7. Vehículo - Motor – Conductor

a. Agregación: Vehículo → Motor

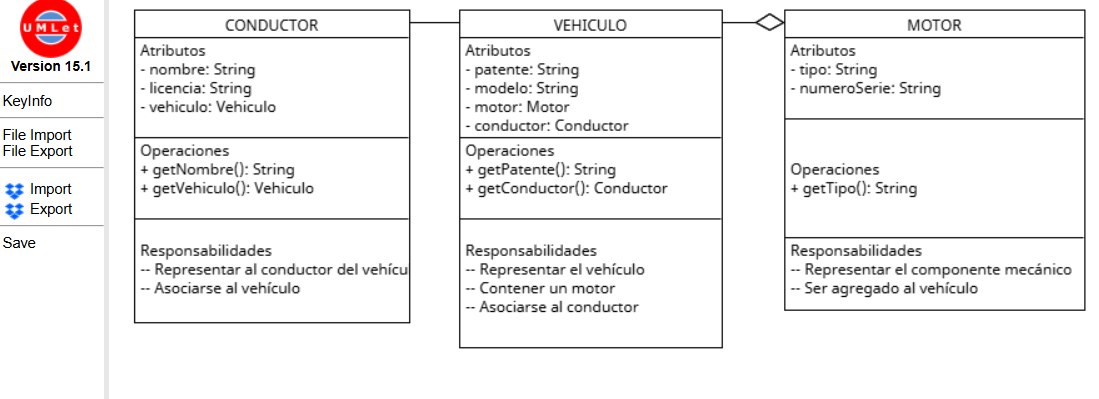
b. Asociación bidireccional: Vehículo ↔ Conductor

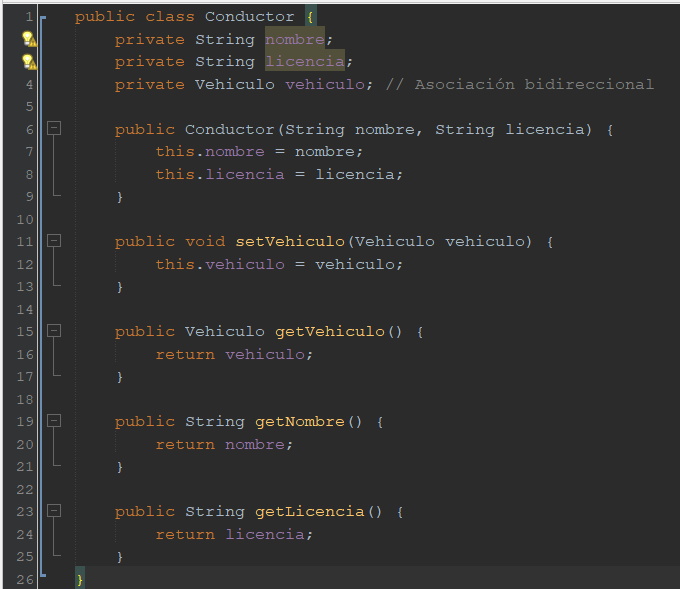
Clases y atributos:

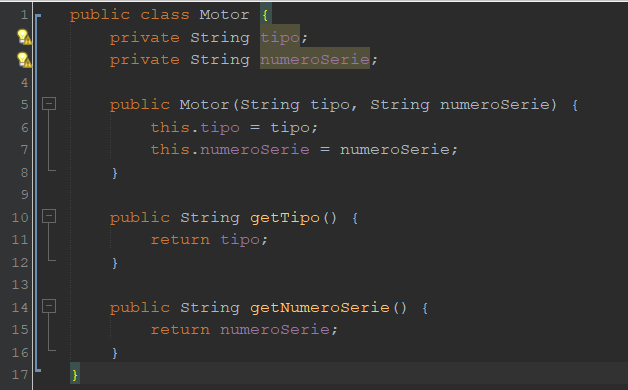
1. Vehículo: patente, modelo
2. Motor: tipo, numeroSerie
3. Conductor: nombre, licencia

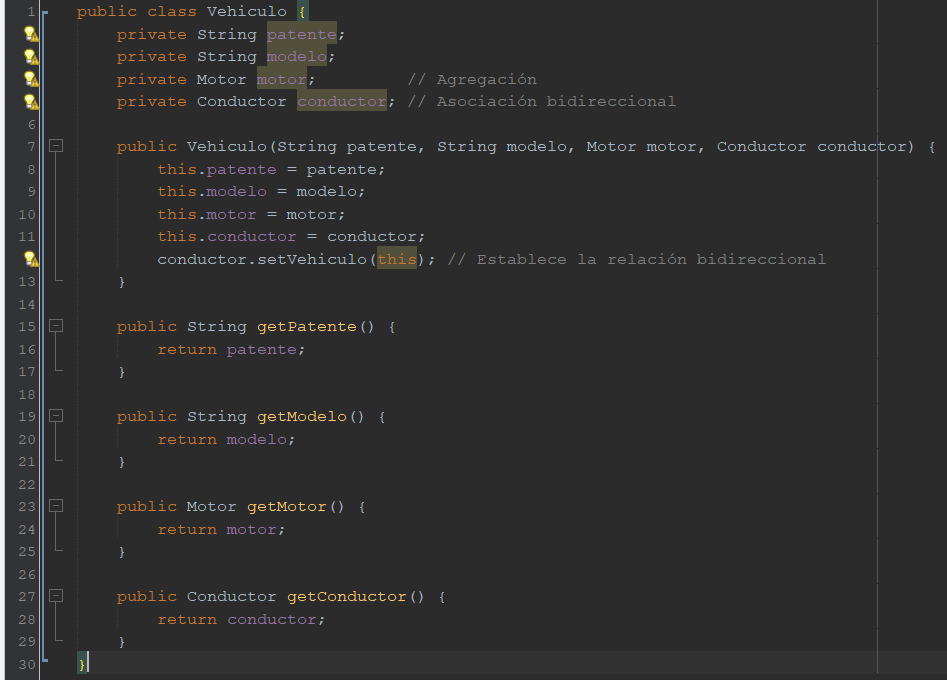
**Agregación**: Vehiculo → Motor (el motor puede existir por separado)

**Asociación bidireccional**: Vehiculo ↔ Conductor (ambos se conocen mutuamente)









8. Documento - FirmaDigital - Usuario

a. Composición: Documento → FirmaDigital

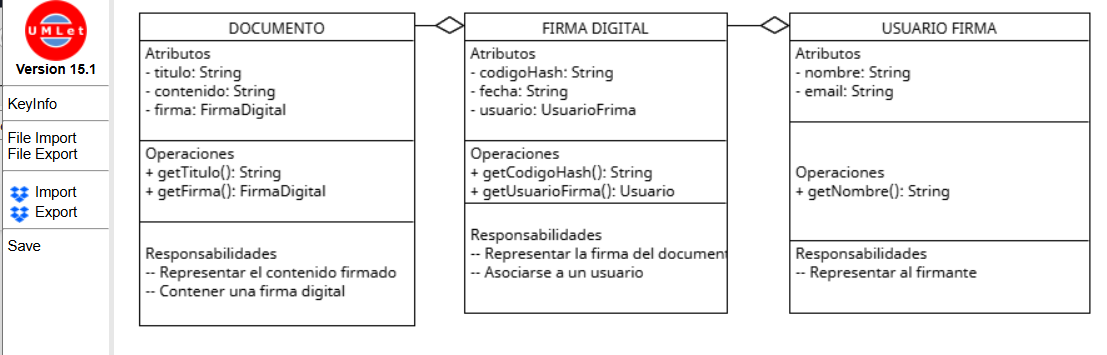
b. Agregación: FirmaDigital → Usuario

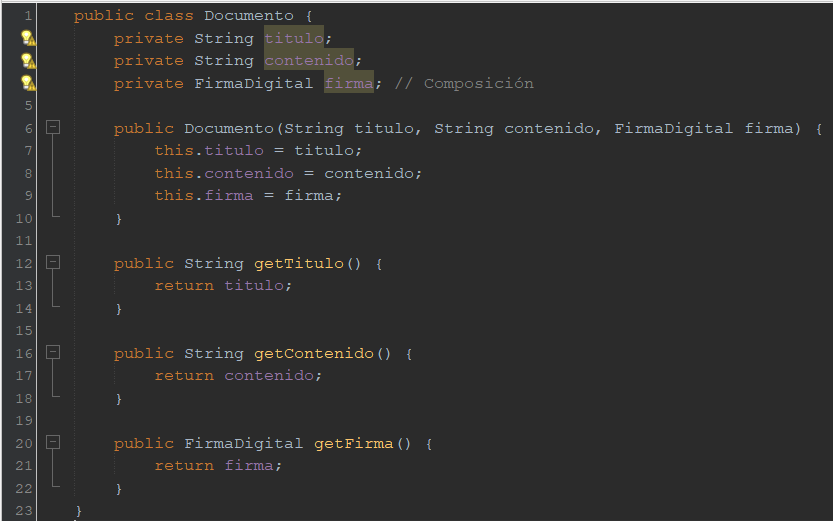
Clases y atributos:

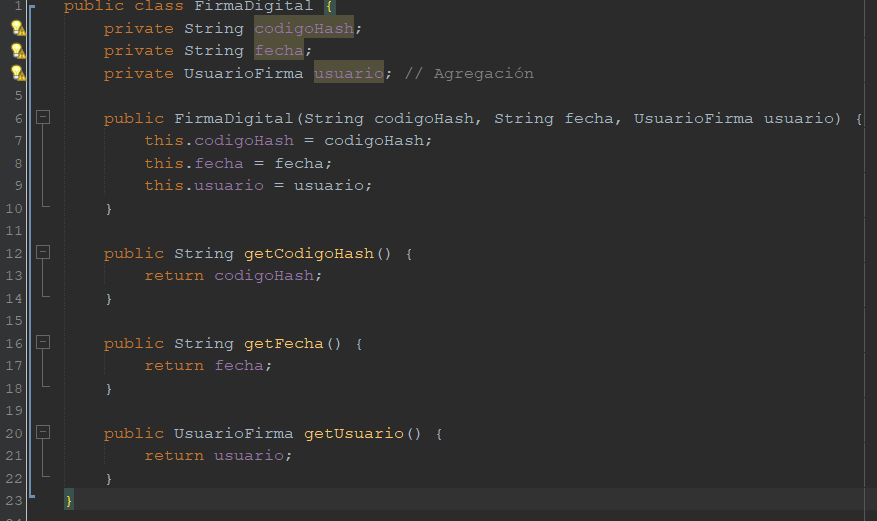
1. Documento: titulo, contenido
2. FirmaDigital: codigoHash, fecha
3. Usuario: nombre, email

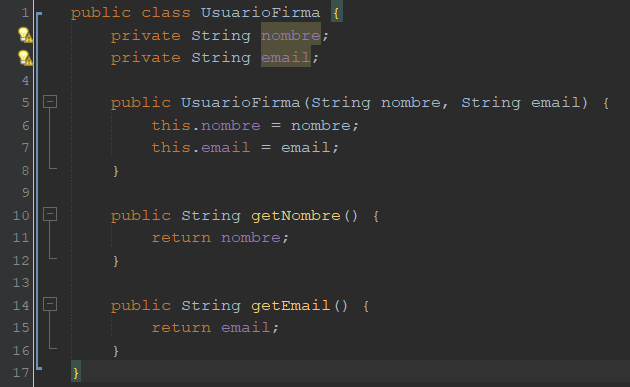
**Composición**: Documento → FirmaDigital (la firma no existe sin el documento)

**Agregación**: FirmaDigital → Usuario (el usuario puede existir por separado)









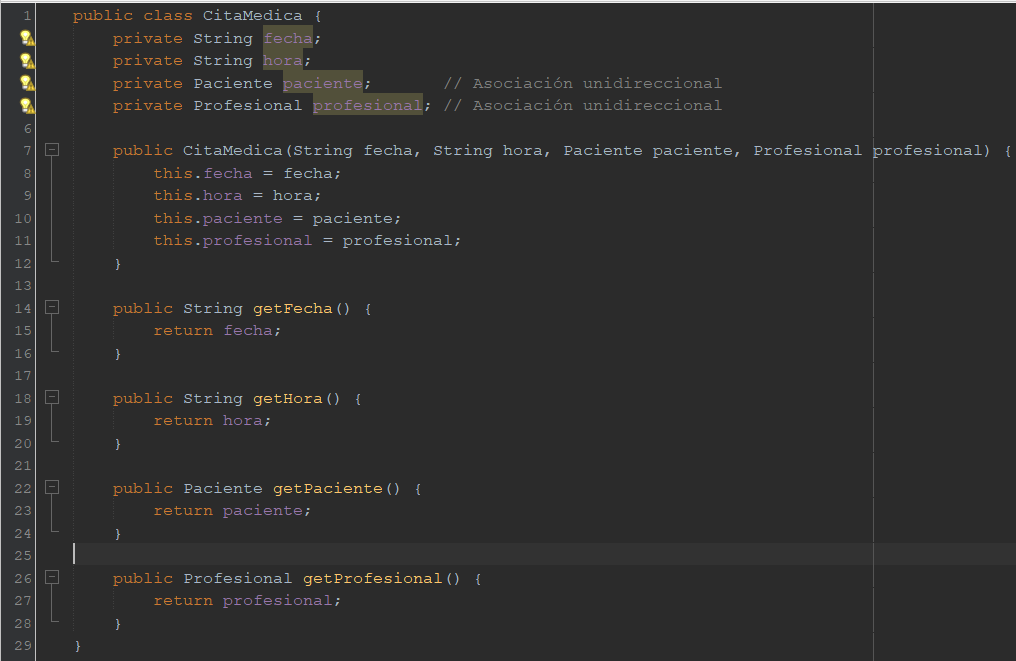
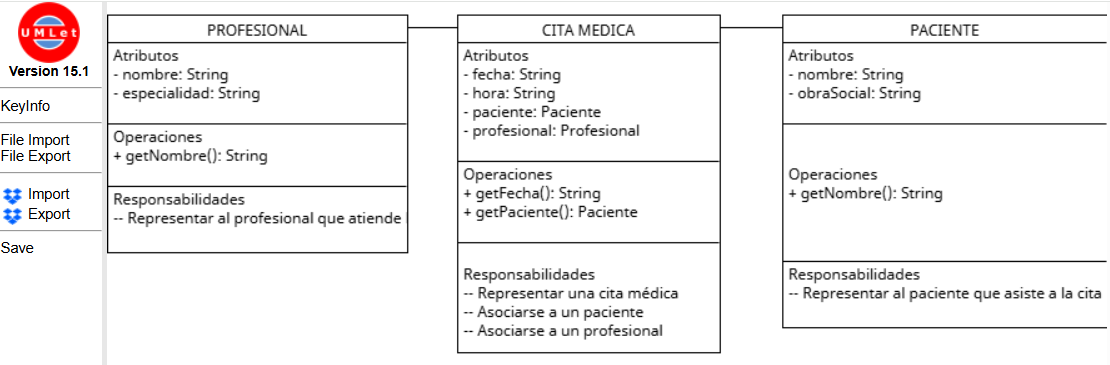
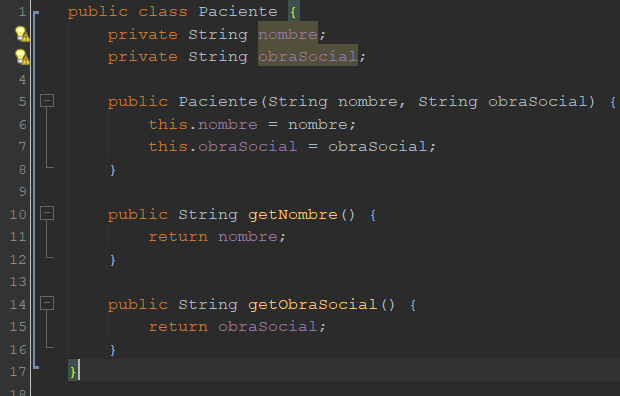
9. CitaMédica - Paciente - Profesional

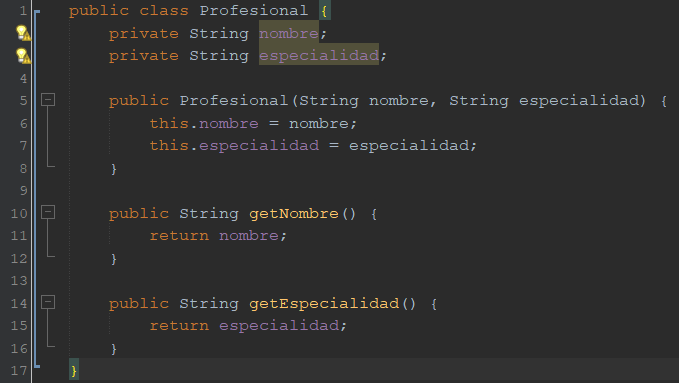
a. Asociación unidireccional: CitaMédica → Paciente,

b. Asociación unidirecciona: CitaMédica → Profesional

Clases y atributos:

1. CitaMédica: fecha, hora
2. Paciente: nombre, obraSocial
3. Profesional: nombre, especialidad

**Asociación unidireccional**: CitaMedica → Paciente y CitaMedica → Profesional  
(la cita conoce al paciente y al profesional, pero ellos no conocen a la cita) 



10. CuentaBancaria - ClaveSeguridad - Titular

a. Composición: CuentaBancaria → ClaveSeguridad

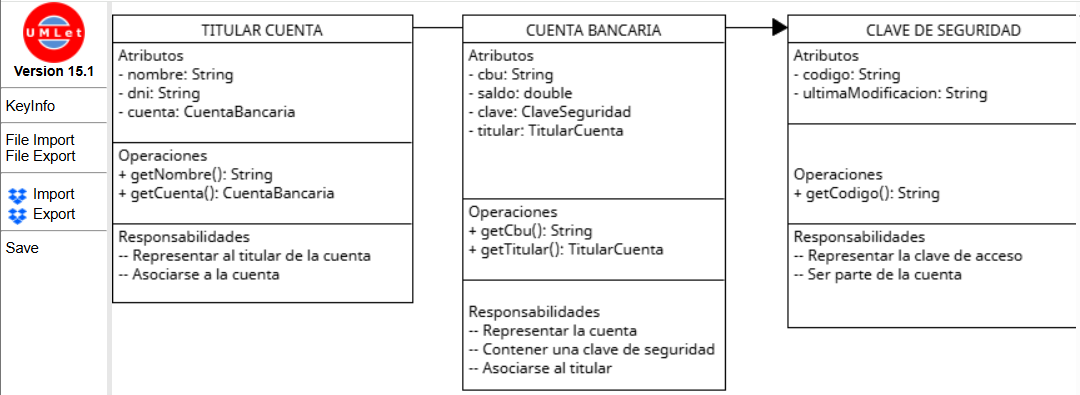
b. Asociación bidireccional: CuentaBancaria ↔ Titular

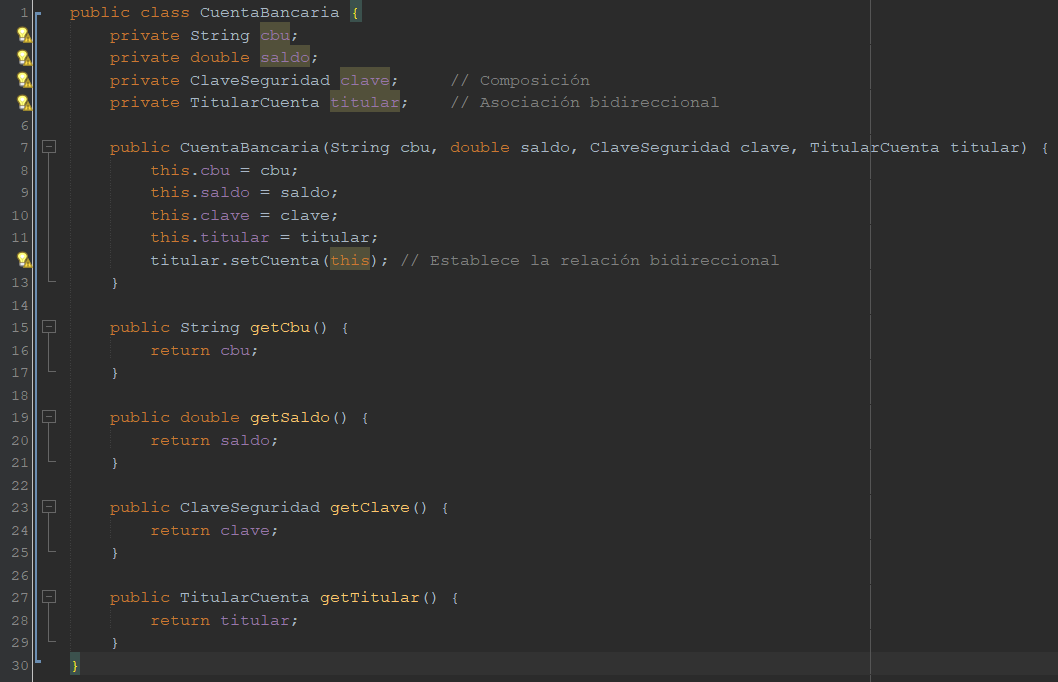
Clases y atributos:

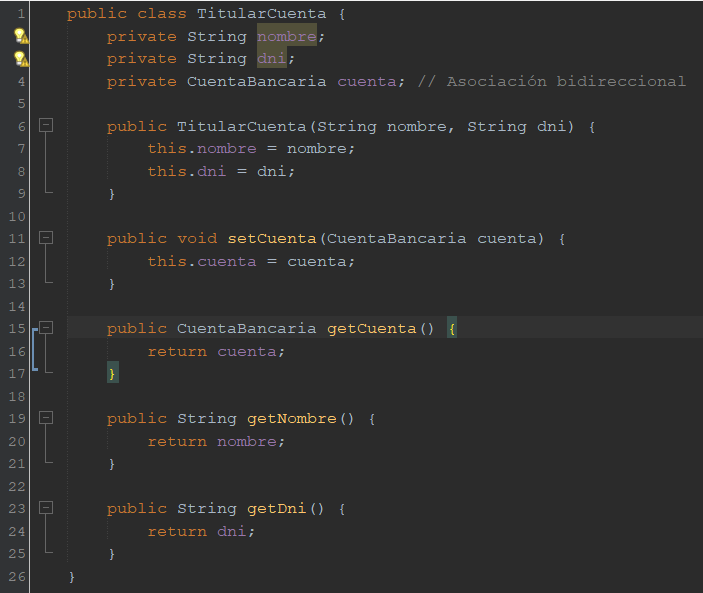
1. CuentaBancaria: cbu, saldo
2. ClaveSeguridad: codigo, ultimaModificacion
3. Titular: nombre, dni.

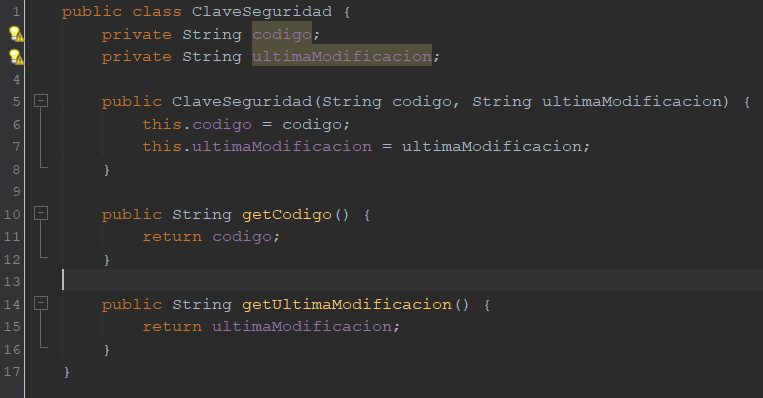
**Composición**: CuentaBancaria → ClaveSeguridad (la clave no existe sin la cuenta)

**Asociación bidireccional**: CuentaBancaria ↔ TitularCuenta (ambos se conocen mutuamente)









**DEPENDENCIA DE USO**

La clase usa otra como parámetro de un método, pero no la guarda como atributo. Ejercicios de Dependencia de Uso

11. Reproductor - Canción - Artista

a. Asociación unidireccional: Canción → Artista

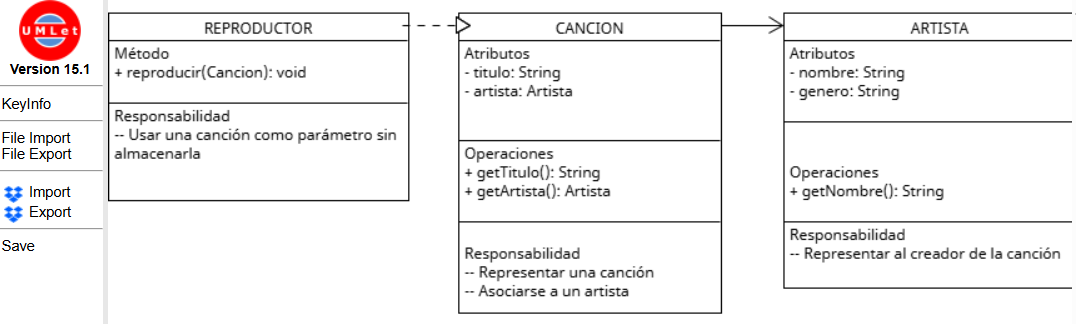
b. Dependencia de uso: Reproductor.reproducir(Cancion)

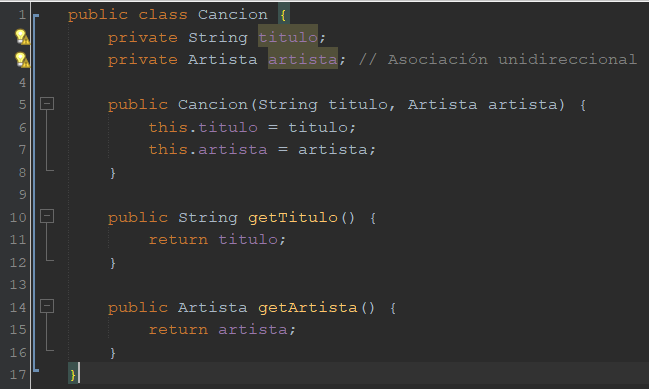
Clases y atributos:

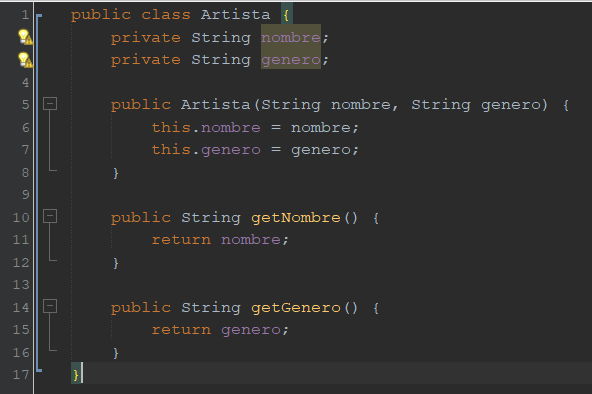
1. Canción: titulo.
2. Artista: nombre, genero.
3. Reproductor->método: void reproducir(Cancion cancion)

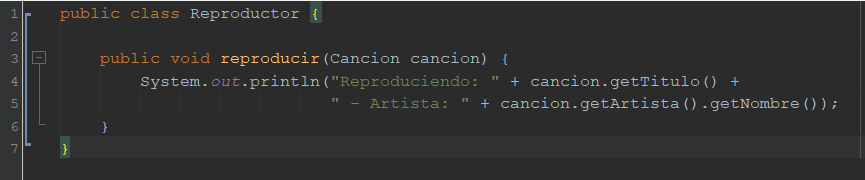
**Asociación unidireccional**: Cancion → Artista (la canción conoce al artista, pero no al revés)

**Dependencia de uso**: Reproductor.reproducir(Cancion) (el reproductor usa la canción como parámetro, sin guardarla como atributo)









12. Impuesto - Contribuyente – Calculadora

a. Asociación unidireccional: Impuesto → Contribuyente

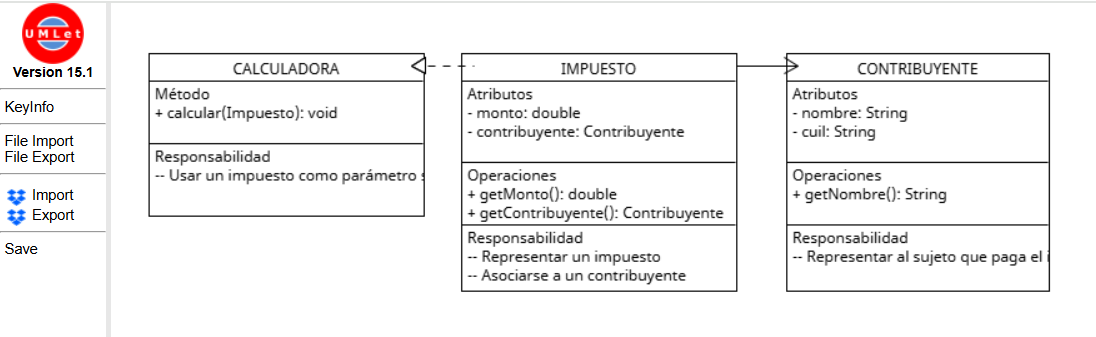
b. Dependencia de uso: Calculadora.calcular(Impuesto)

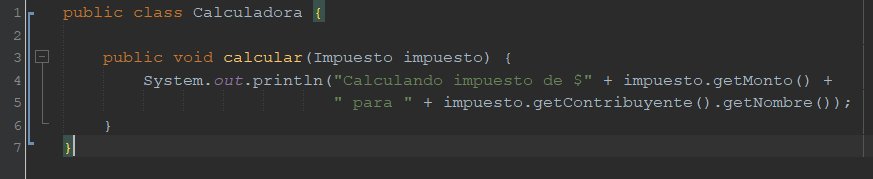
Clases y atributos:

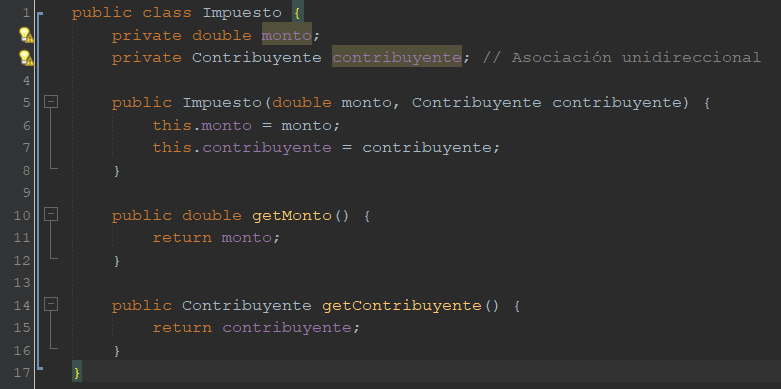
1. Impuesto: monto.
2. Contribuyente: nombre, cuil
3. Calculadora->método: void calcular(Impuesto impuesto)

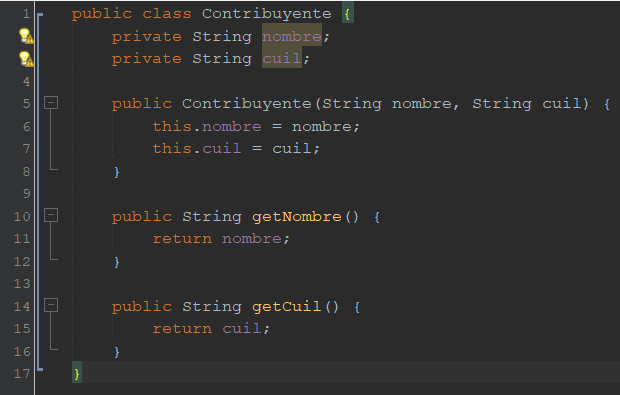
**Asociación unidireccional**: Impuesto → Contribuyente  
(el impuesto conoce al contribuyente, pero no al revés)

**Dependencia de uso**: Calculadora.calcular(Impuesto)  
(la calculadora usa el impuesto como parámetro, sin guardarlo como atributo)









DEPENDENCIA DE CREACIÓN

La clase crea otra dentro de un método, pero no la conserva como atributo..

Ejercicios de Dependencia de Creación

13. GeneradorQR - Usuario - CódigoQR

a. Asociación unidireccional: CódigoQR → Usuario

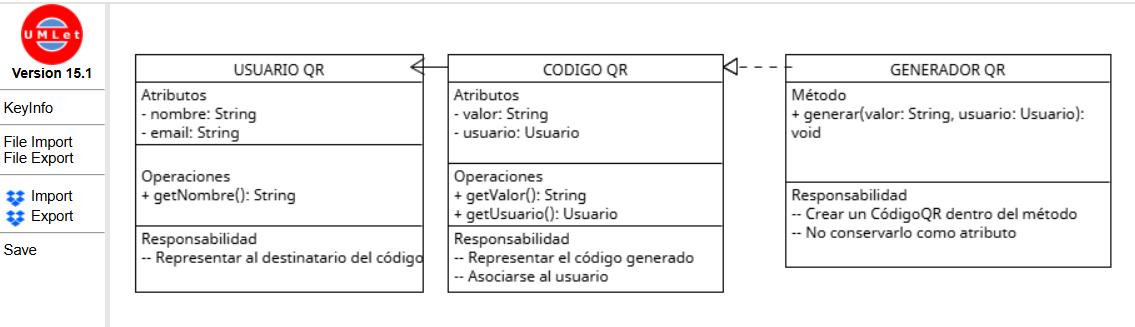
b. Dependencia de creación: GeneradorQR.generar(String, Usuario)

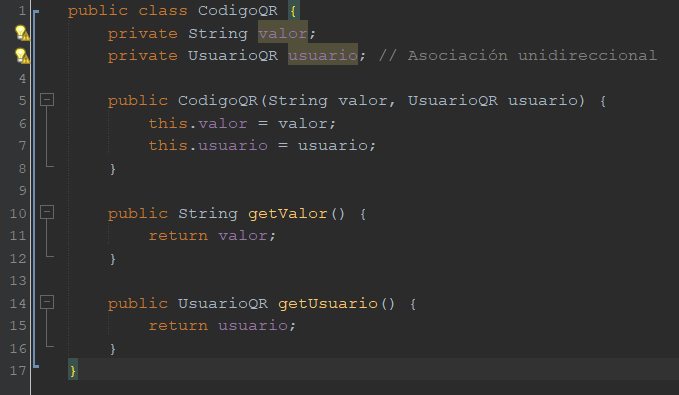
Clases y atributos:

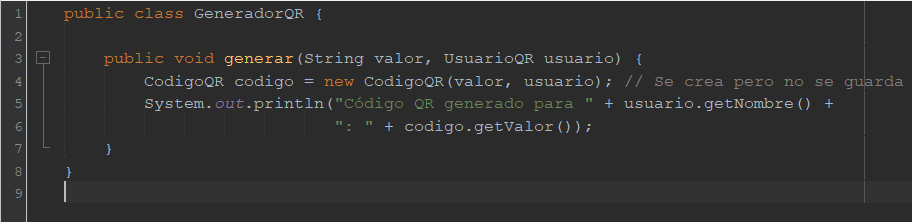
1. CodigoQR: valor.
2. Usuario: nombre, email.
3. GeneradorQR->método: void generar(String valor, Usuario usuario)

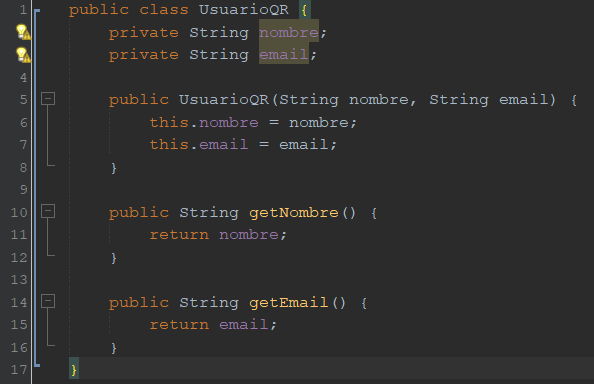
**Asociación unidireccional**: CódigoQR → Usuario  
La clase CódigoQR mantiene una referencia al objeto Usuario, lo que significa que conoce quién es el destinatario del código. Sin embargo, el Usuario no conoce al CódigoQR, por lo tanto la relación es unidireccional.

**Dependencia de creación**: GeneradorQR → CódigoQR  
La clase GeneradorQR **crea una instancia de CódigoQR dentro del método generar(...)**, la utiliza en ese momento (por ejemplo, para mostrar el resultado), pero **no la guarda como atributo**. Esto implica que la relación entre GeneradorQR y CódigoQR es temporal y funcional, no estructural.









14. EditorVideo - Proyecto - Render

a. Asociación unidireccional: Render → Proyecto

b. Dependencia de creación: EditorVideo.exportar(String, Proyecto)

Clases y atributos:

1. Render: formato.
2. Proyecto: nombre, duracionMin.
3. EditorVideo->método: void exportar(String formato, Proyecto proyecto)

**Asociación unidireccional**: Render → Proyecto  
La clase Render mantiene una referencia al proyecto que exporta, pero el proyecto no conoce al render.

**Dependencia de creación**: EditorVideo → Render  
La clase EditorVideo crea una instancia de Render dentro del método exportar(), la utiliza en ese momento, pero no la guarda como atributo. La relación es temporal y funcional.

