TRABAJO PRACTICO 5 – UML

PROGRAMACION 2

ALUMNO – EMILIANO FABIAN JARA

ACTIVIDADES:

1) Pasaporte – Foto – Titular

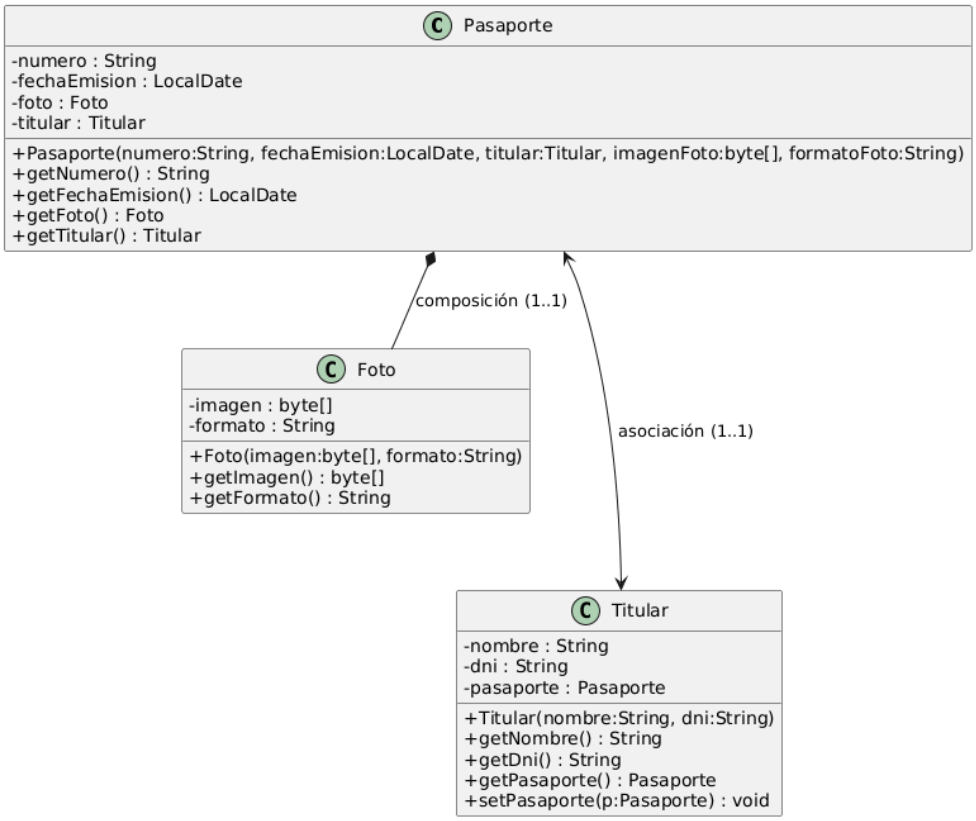
Objetivo del ejercicio

- **Composición (1 a 1)**: Pasaporte → Foto
  La **Foto** existe *solo* dentro del **Pasaporte** (vida dependiente).
- **Asociación bidireccional (1 a 1)**: Pasaporte ↔ Titular
  Tanto **Pasaporte** como **Titular** se referencian mutuamente (cada uno conoce al otro).

Atributos (según enunciado)

- **Pasaporte**: numero, fechaEmision
- **Foto**: imagen, formato
- **Titular**: nombre, dni

DIAGRAMA:



La asociación Pasaporte ↔ Titular es **bidireccional**: ambos tienen referencia entre sí. Por simplicidad, lo normal es **setear** esta relación en el constructor de Pasaporte y/o un setPasaporte en Titular para mantener la **invariante 1 a 1**.

Justificación del diseño

- **Composición Pasaporte→Foto**: la Foto no tiene sentido fuera del Pasaporte; por eso se **crea dentro** y no se expone ningún setter para reemplazarla. Si el Pasaporte se pierde, la Foto también.
- **Asociación bidireccional Pasaporte↔Titular**: ambos se **referencian**; se asegura la **invariante 1 a 1** evitando múltiples pasaportes por titular mediante una verificación en setPasaporte.
- **Encapsulamiento** y uso de **copias defensivas** para el byte[] de la imagen, evitando mutaciones externas.

2) Celular – Batería – Usuario

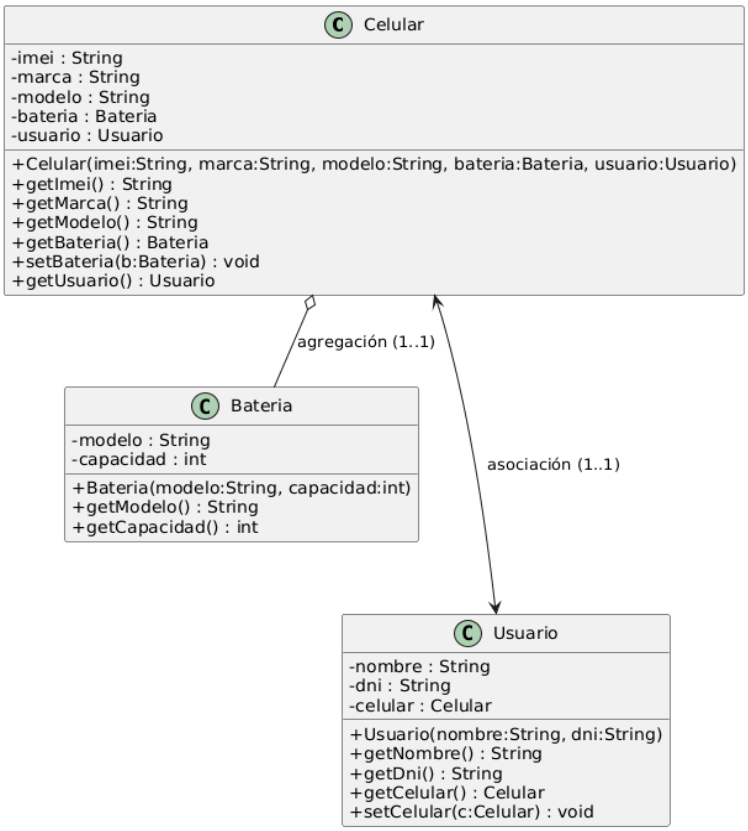
Relaciones requeridas

- **Agregación (1 a 1)**: Celular → Batería
  La **Batería** puede existir independientemente del **Celular** (no depende de su ciclo de vida).
- **Asociación bidireccional (1 a 1)**: Celular ↔ Usuario
  Celular y **Usuario** se referencian mutuamente.

Atributos (según enunciado)

- **Celular**: imei, marca, modelo
- **Batería**: modelo, capacidad
- **Usuario**: nombre, dni

DIAGRAMA :



- La **asociación bidireccional** garantiza que **ambos lados** puedan navegar la relación (cada uno guarda referencia del otro).

3) Libro – Autor – Editorial

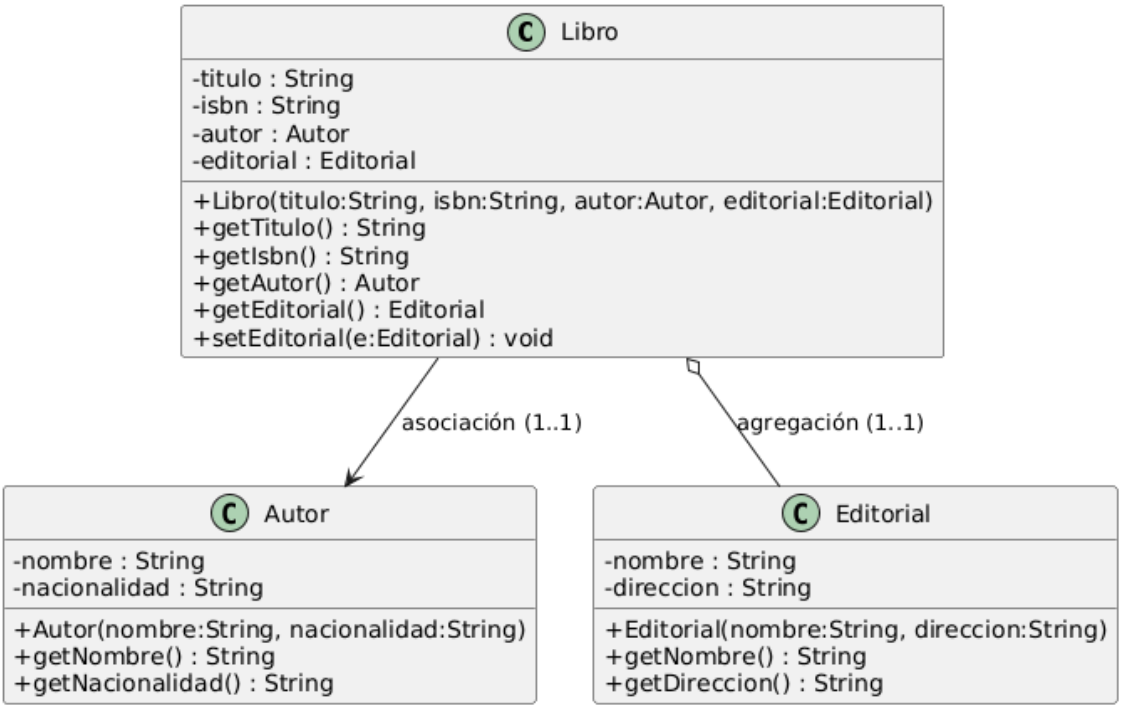
Relaciones requeridas (según enunciado)

- **Asociación unidireccional (1 a 1)**: Libro → Autor
  El **Libro** conoce a su **Autor**, pero el **Autor** no conoce al **Libro**.
- **Agregación (1 a 1)**: Libro → Editorial
  El **Libro** "tiene" una **Editorial**, pero **Editorial** puede existir independientemente y puede cambiarse.

Atributos

- **Libro**: titulo, isbn
- **Autor**: nombre, nacionalidad
- **Editorial**: nombre, dirección

Diagrama 3:



asociación unidireccional: solo Libro referencia a Autor.

agregación: Libro *tiene* una Editorial que puede existir por fuera y reemplazarse.

4) TarjetaDeCrédito – Cliente – Banco

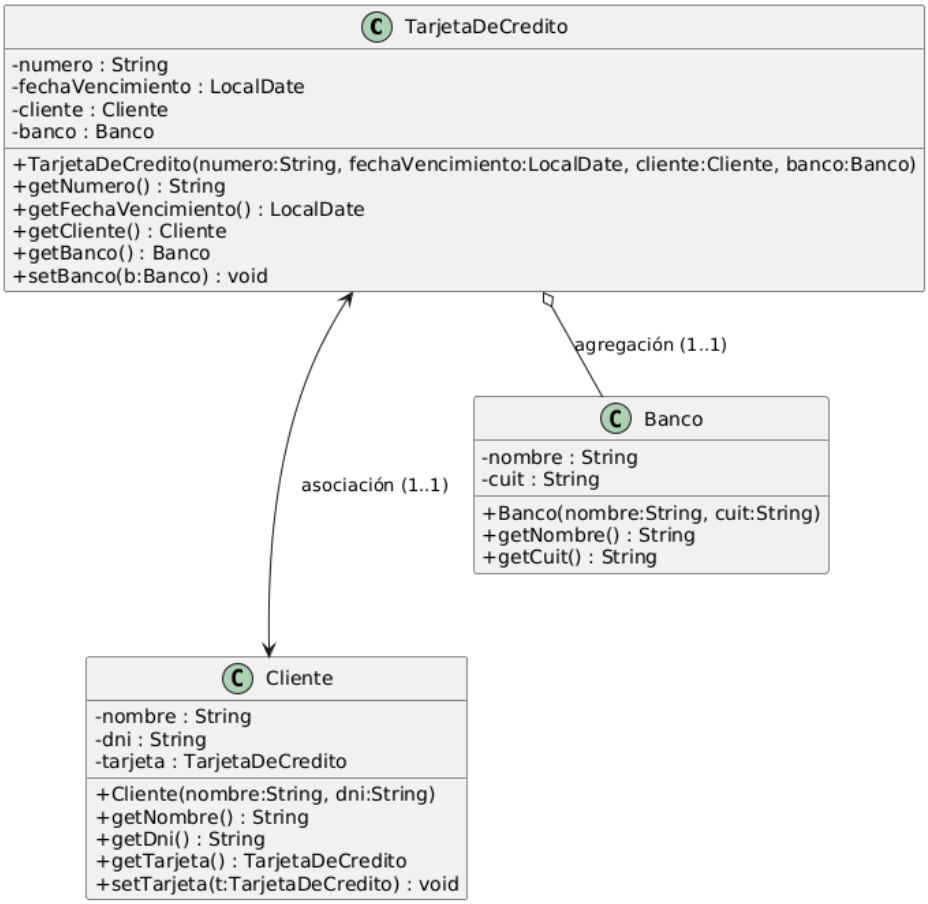
Relaciones requeridas

- **Asociación bidireccional (1 a 1)**: TarjetaDeCrédito ↔ Cliente
  Ambos lados se referencian y se mantiene la invariante 1 a 1.
- **Agregación (1 a 1)**: TarjetaDeCrédito → Banco
  La tarjeta *tiene* un banco, pero el banco puede existir independientemente y puede cambiarse.

Atributos

- **TarjetaDeCrédito**: numero, fechaVencimiento
- **Cliente**: nombre, dni
- **Banco**: nombre, cuit

Diagrama



- La flecha doble sin diamantes indica **asociación bidireccional**: ambos guardan referencia y pueden navegar la relación.
- El **diamante blanco** en TarjetaDeCredito o -- Banco indica **agregación**: la tarjeta puede cambiar de banco y el banco existe por fuera.

5) Computadora – PlacaMadre – Propietario

Relaciones requeridas

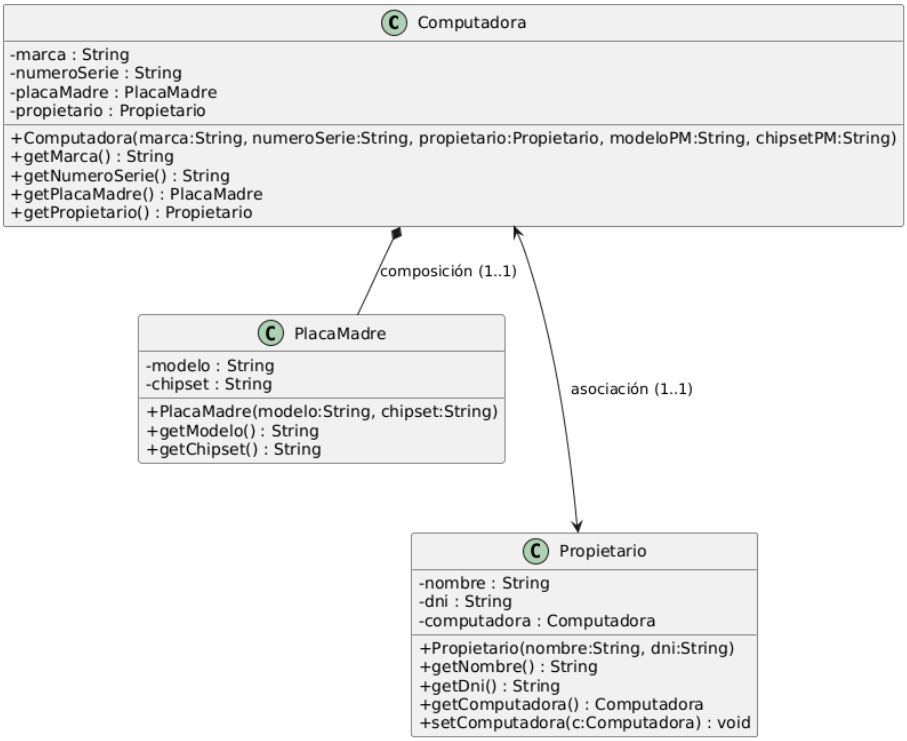
- **Composición (1 a 1)**: Computadora → PlacaMadre
  La **PlacaMadre** *depende* del ciclo de vida de la **Computadora**. La crea y la posee la Computadora.
- **Asociación bidireccional (1 a 1)**: Computadora ↔ Propietario
  Ambos lados se referencian mutuamente.

Atributos

- **Computadora**: marca, numeroSerie

- **PlacaMadre**: modelo, chipset
- **Propietario**: nombre, dni

  Diagrama:



- *-- con **diamante negro** representa **composición**: PlacaMadre se crea dentro de Computadora y no vive por fuera.
- La flecha doble indica **asociación bidireccional**: ambos guardan referencia y pueden navegar la relación.

6) Reserva – Cliente – Mesa

Relaciones requeridas (según enunciado)

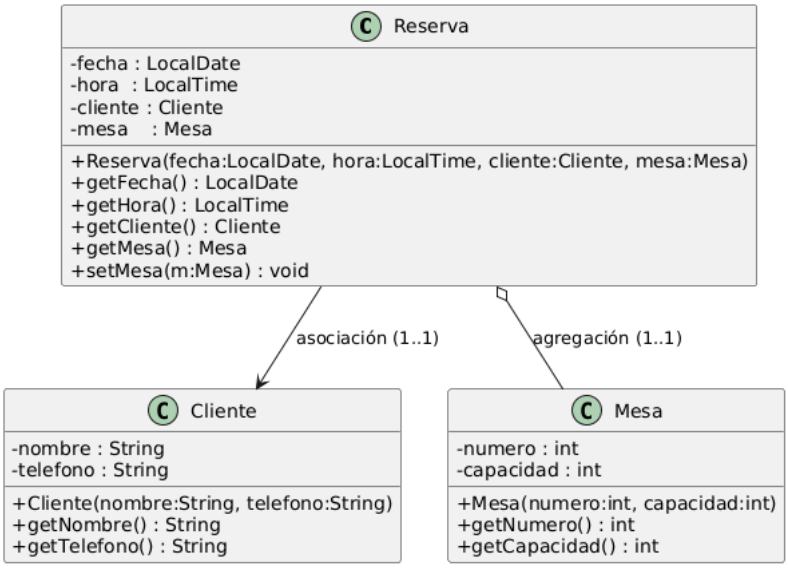
- **Asociación unidireccional (1 a 1)**: Reserva → Cliente
  La **Reserva** conoce al **Cliente**, pero el **Cliente** no conoce la **Reserva**.
- **Agregación (1 a 1)**: Reserva → Mesa
  La **Reserva** "tiene" una **Mesa**, pero la **Mesa** puede existir independientemente y puede cambiarse.

Atributos

- **Reserva**: fecha, hora

- **Cliente**: nombre, telefono
- **Mesa**: numero, capacidad

Diagrama:



Notas UML

- La flecha (-->) indica **asociación unidireccional**: solo Reserva referencia a Cliente.
- El **diamante blanco** (o--) indica **agregación**: Reserva *tiene* una Mesa que puede existir por fuera y **cambiarse**.

7) Vehículo – Motor – Conductor

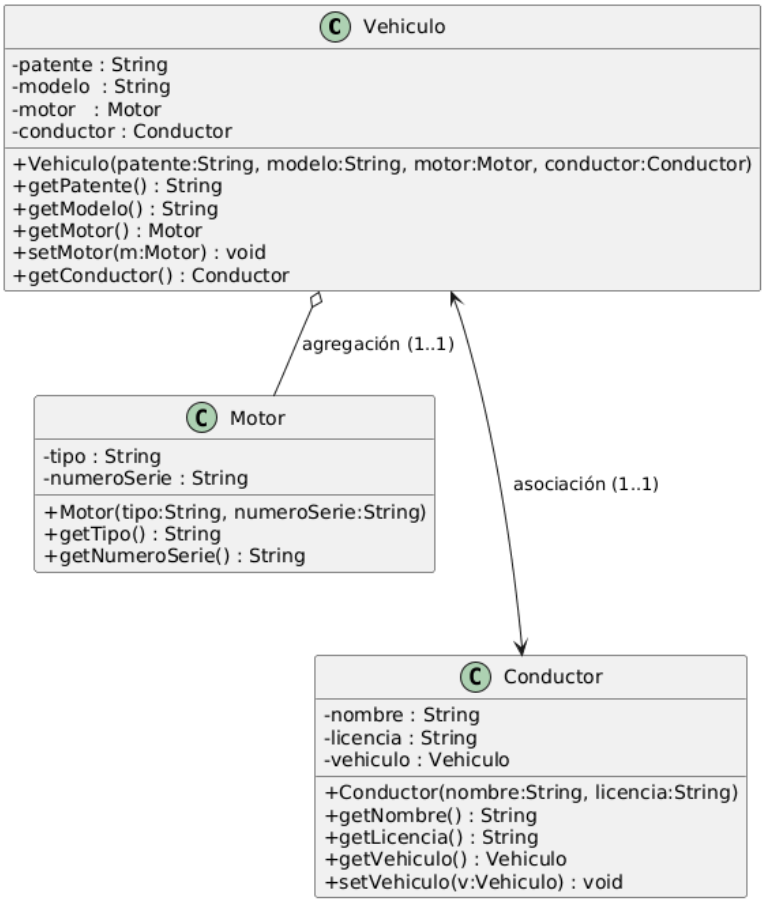
Relaciones requeridas

- **Agregación (1 a 1)**: Vehículo → Motor
  El **Vehículo** "tiene" un **Motor**, pero el Motor puede existir por fuera y puede reemplazarse.
- **Asociación bidireccional (1 a 1)**: Vehículo ↔ Conductor
  Ambos lados se referencian mutuamente (cada uno conoce al otro).

Atributos

- **Vehículo**: patente, modelo
- **Motor**: tipo, numeroSerie
- **Conductor**: nombre, licencia

Diagrama :



Notas UML

- El **diamante blanco** (o--) representa **agregación**: Motor puede existir independientemente y **puede cambiarse** con setMotor.
- La relación Vehiculo ↔ Conductor es **bidireccional**: ambos tienen referencia; se suele establecer desde el constructor de Vehiculo llamando a conductor.setVehiculo(this).

8) Documento – FirmaDigital – Usuario

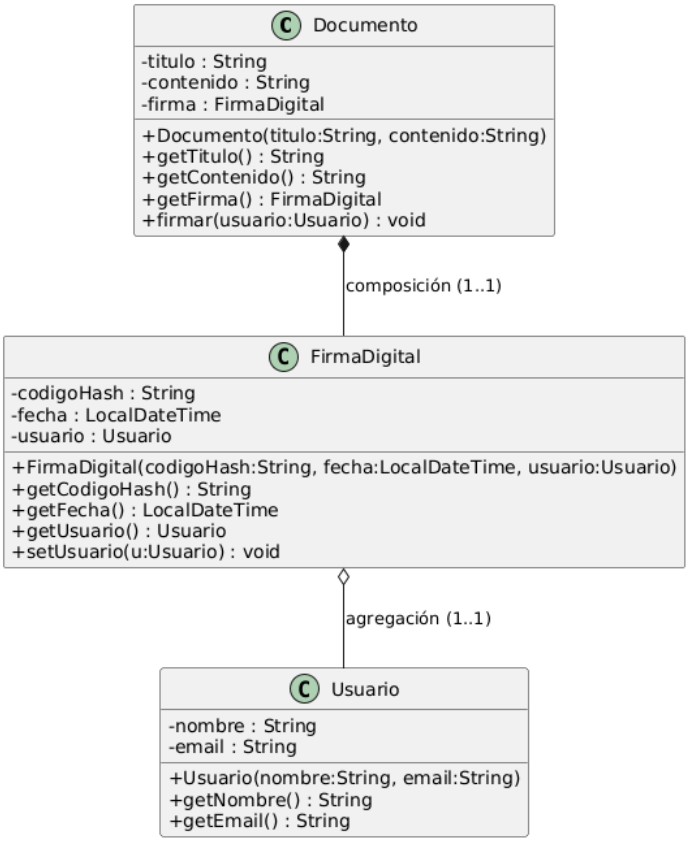
Relaciones requeridas

- **Composición (1 a 1)**: Documento → FirmaDigital
  La **FirmaDigital** depende del ciclo de vida del **Documento**. El Documento la crea y la posee.
- **Agregación (1 a 1)**: FirmaDigital → Usuario
  La **FirmaDigital** "tiene" un **Usuario** (firmante), pero el **Usuario** puede existir independientemente y puede cambiarse.

Atributos (según enunciado)

- **Documento**: titulo, contenido
- **FirmaDigital**: codigoHash, fecha
- **Usuario**: nombre, email

DIAGRAMA:



- con **diamante negro** representa **composición**: FirmaDigital se crea desde Documento y **no vive por fuera**.
- con **diamante blanco** representa **agregación**: FirmaDigital *tiene* un Usuario que puede existir independientemente y **puede cambiarse** (p. ej., reasignar firmante antes de emitir la versión final).

9) CitaMédica – Paciente – Profesional

Relaciones requeridas (1 a 1)

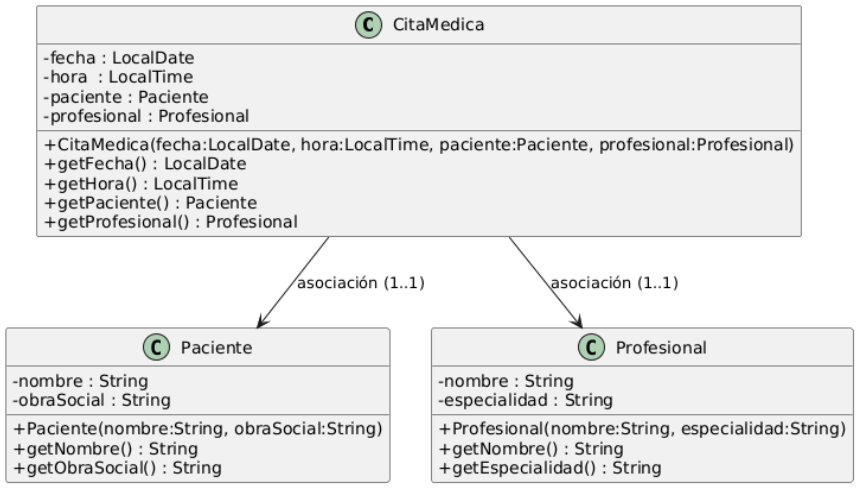
- **Asociación unidireccional**: CitaMédica → Paciente
  La **CitaMédica** conoce al **Paciente**; el **Paciente** no conoce la **CitaMédica**.

- **Asociación unidireccional**: CitaMédica → Profesional
  La **CitaMédica** conoce al **Profesional**; el **Profesional** no conoce la **CitaMédica**.

  Atributos (según enunciado)

- **CitaMédica**: fecha, hora
- **Paciente**: nombre, obraSocial
- **Profesional**: nombre, especialidad

DIAGRAMA



- Las flechas (-->) indican **unidireccionalidad**: **solo** CitaMedica referencia a Paciente y Profesional.
- Se mantiene el **cardinal 1..1** en ambos vínculos según el enunciado.

10) CuentaBancaria – ClaveSeguridad – Titular

Relaciones requeridas

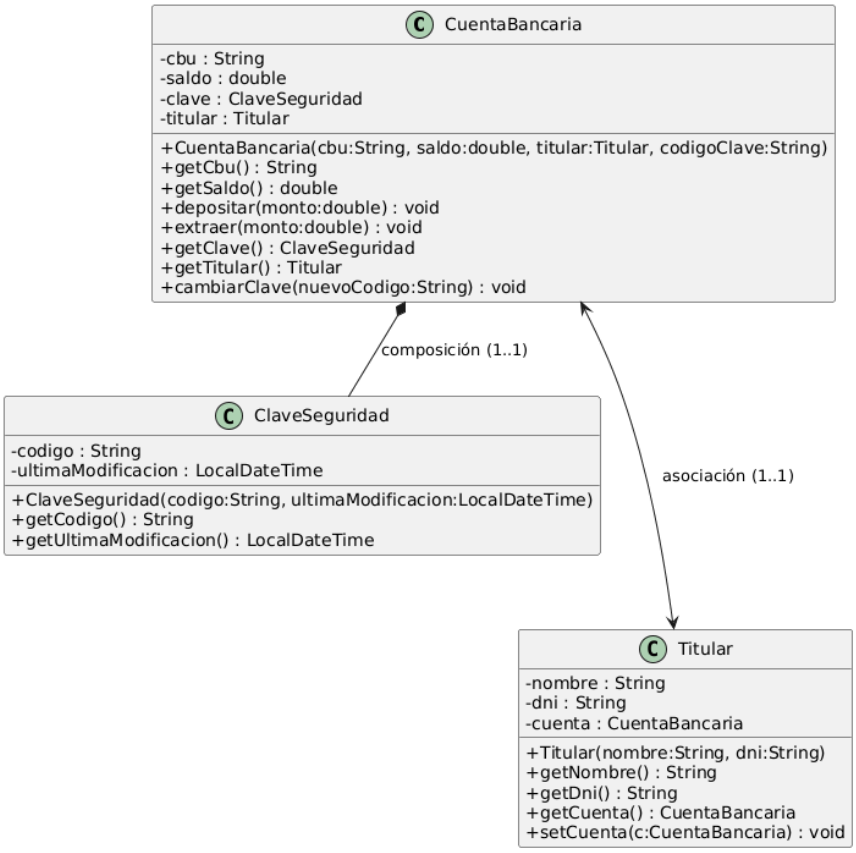
- **Composición (1 a 1)**: CuentaBancaria → ClaveSeguridad
  La **ClaveSeguridad** depende del ciclo de vida de la **CuentaBancaria**; la cuenta la crea y la administra.
- **Asociación bidireccional (1 a 1)**: CuentaBancaria ↔ Titular
  Ambos lados se referencian mutuamente, manteniendo la invariante 1 a 1.

  Atributos

- **CuentaBancaria**: cbu, saldo

- **ClaveSeguridad**: codigo, ultimaModificacion
- **Titular**: nombre, dni

  Diagrama



- con **diamante negro** representa **composición**: ClaveSeguridad se crea y se reemplaza **desde** CuentaBancaria.
- La flecha doble indica **asociación bidireccional**: CuentaBancaria y Titular guardan referencia mutua.

11) Reproductor – Canción – Artista

Relaciones requeridas

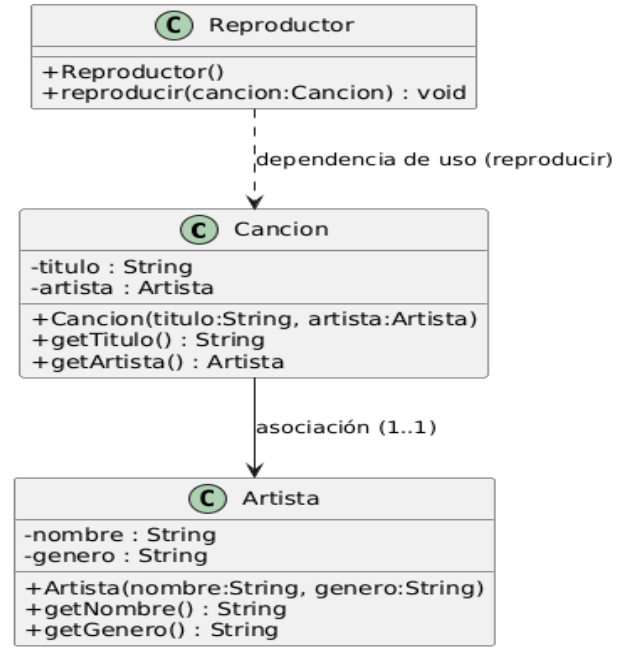
- **Asociación unidireccional (1 a 1)**: Cancion → Artista
  La **Cancion** conoce al **Artista**, pero el **Artista** no conoce a la Cancion.

- **Dependencia de uso**: método Reproductor.reproducir(Cancion) **Reproductor usa** una **Cancion como parámetro** y **no** la guarda como atributo.

Atributos

- **Cancion**: titulo
- **Artista**: nombre, genero
- **Reproductor**: método void reproducir(Cancion cancion)

Diagrama



- La flecha sólida --> marca **asociación unidireccional** entre Cancion y Artista.
- La flecha **discontinua** ..> representa **dependencia**: Reproductor **usa** Cancion **solo** en el método reproducir.

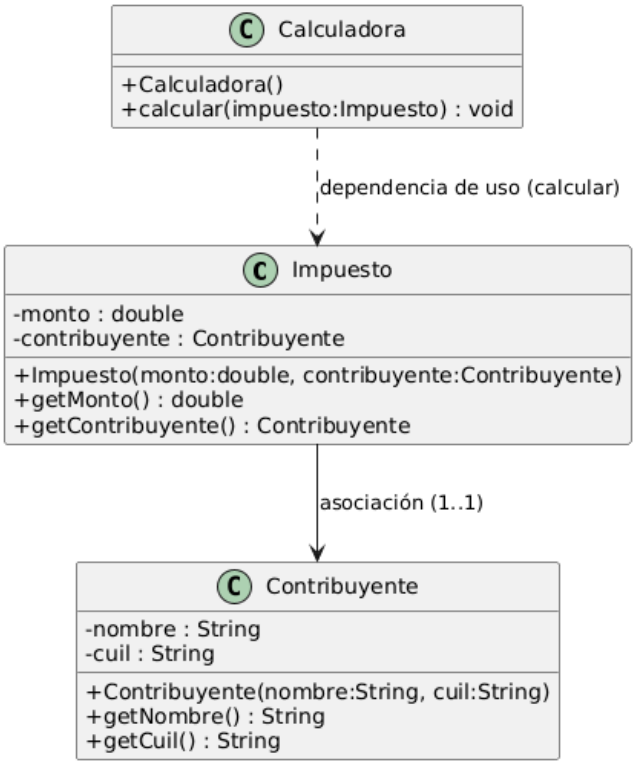
12) Impuesto – Contribuyente – Calculadora

Relaciones requeridas

- **Asociación unidireccional (1 a 1)**: Impuesto → Contribuyente
  El **Impuesto** conoce al **Contribuyente**, pero el **Contribuyente** no conoce al Impuesto.
- **Dependencia de uso**: Calculadora.calcular(Impuesto)
  Calculadora usa un **Impuesto** como **parámetro de método** y **no** lo guarda como atributo.

  Atributos (según enunciado)
- **Impuesto**: monto
- **Contribuyente**: nombre, cuil
- **Calculadora**: método void calcular(Impuesto impuesto)

DIAGRAMA:



- La flecha sólida --> marca **asociación unidireccional** entre Impuesto y Contribuyente.
- La flecha **discontinua** ..> representa **dependencia**: Calculadora **usa** Impuesto **solo** dentro del método calcular.

13) GeneradorQR – Usuario – CódigoQR

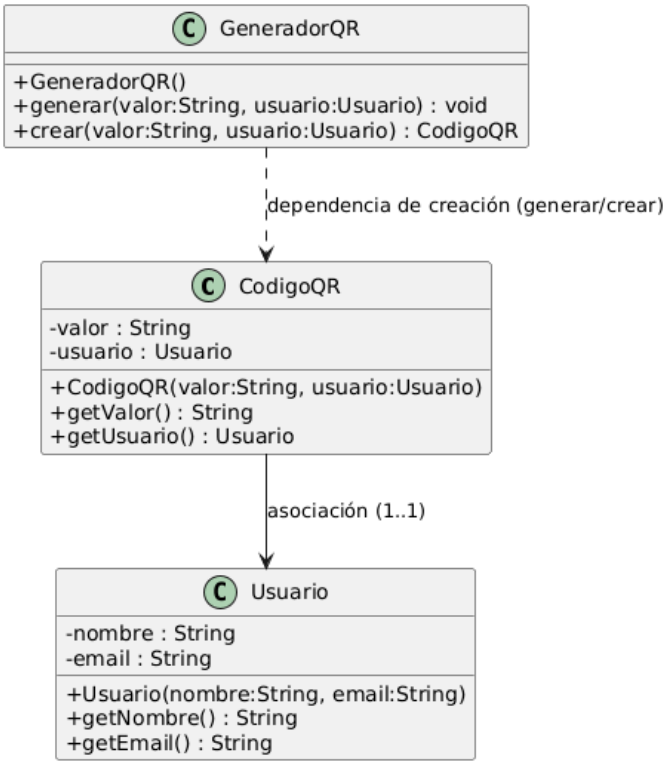
Relaciones requeridas

- **Asociación unidireccional (1 a 1)**: CodigoQR → Usuario
  El **CódigoQR** conoce al **Usuario** (dueño del código), pero el **Usuario** no conoce al **CódigoQR**.
- **Dependencia de creación**: GeneradorQR.generar(String valor, Usuario usuario)
  GeneradorQR crea un **CódigoQR** dentro de un método, **sin** guardarlo como atributo.

Atributos (según enunciado)

- **CodigoQR**: valor
- **Usuario**: nombre, email
- **GeneradorQR**: método void generar(String valor, Usuario usuario)

Diagrama:



- La flecha sólida --> en CodigoQR --> Usuario indica **unidireccionalidad**: solo CodigoQR referencia a Usuario.
- La flecha discontinua ..> indica **dependencia**: GeneradorQR **crea** CodigoQR en un método, sin conservarlo como atributo (no hay relación de atributo entre ellos).

14) EditorVideo – Proyecto – Render

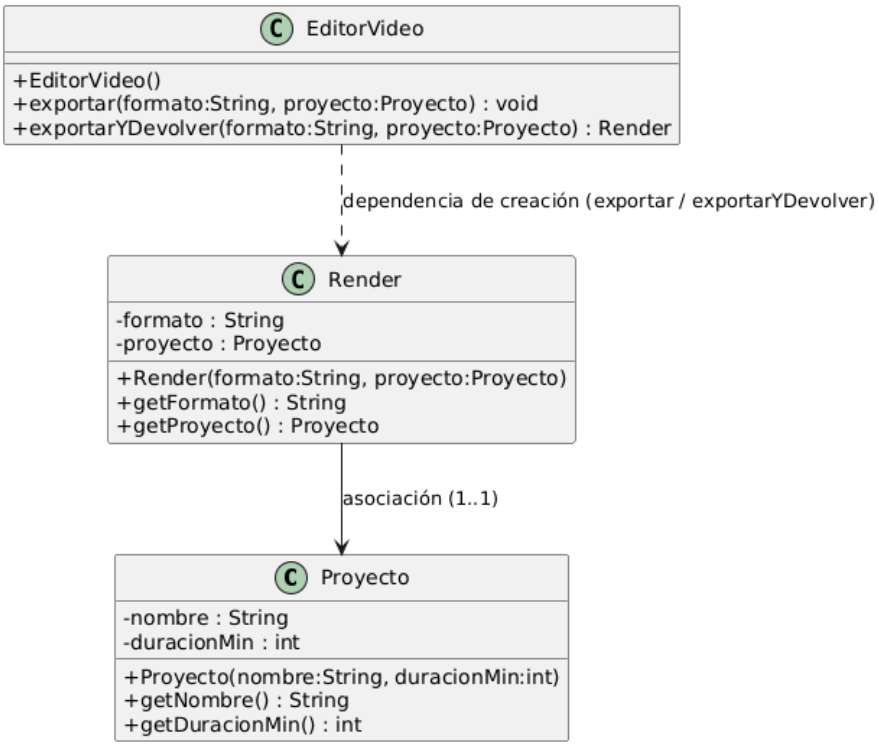
Relaciones requeridas

- **Asociación unidireccional (1 a 1)**: Render → Proyecto
  El **Render** conoce al **Proyecto**, pero el **Proyecto** no conoce al **Render**.
- **Dependencia de creación**: EditorVideo.exportar(String formato, Proyecto proyecto)
  EditorVideo crea un **Render** dentro de un método, **sin** guardarlo como atributo (no hay relación de atributo entre EditorVideo y Render).

Atributos (según enunciado)

- **Render**: formato
- **Proyecto**: nombre, duracionMin
- **EditorVideo**: método void exportar(String formato, Proyecto proyecto)

diagrama



- La flecha sólida --> indica **asociación unidireccional**: Render referencia a Proyecto.
- La flecha discontinua ..> marca **dependencia**: EditorVideo crea Render **dentro** del método y **no** lo conserva como atributo.