TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

PROGRAMACIÓN II

Trabajo Práctico 5: UML y Relaciones

Autor: IACONO, Federico
Comisión 6
Link repositorio: UML

Correcciones implementadas:

- **En las asociaciones simples, la relación entre objetos se establece ahora mediante el uso de Setters.**
- **En las asociaciones bidireccionales ahora también sucede lo mismo. La relación se establece mediante Setters en ambas clases, e incluyen una condición que evita un bucle infinito al momento de vincular los objetos entre sí.**
- **Ambos tipos de relaciones ahora no se establecen mediante el constructor.**
- **Corrección del diagrama de clase del ejercicio 10.**

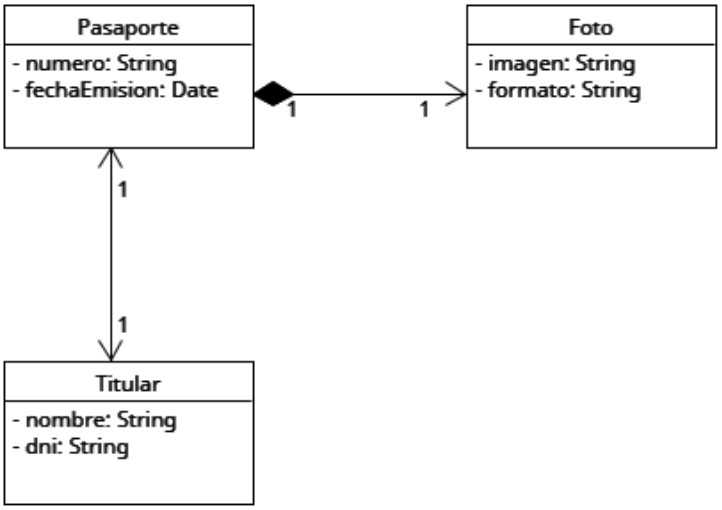
CASO PRÁCTICO

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

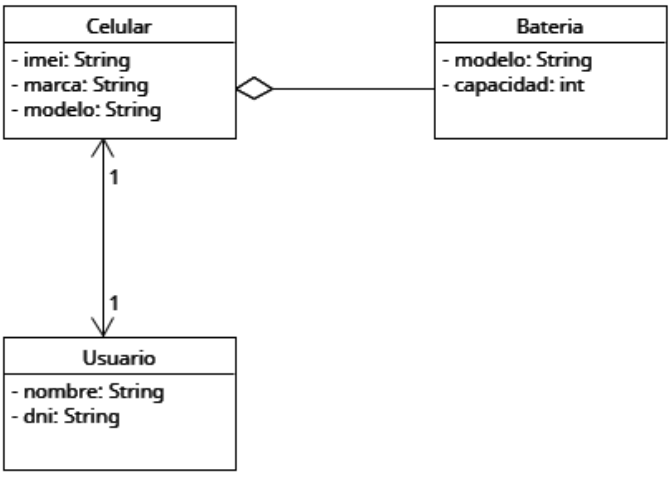


Relaciones:

- Pasaporte → Foto: Se trata de una composición. La Foto no puede existir sin el Pasaporte. Cuando se destruye el objeto Pasaporte, la Foto también se destruye. La dirección es unidireccional desde Pasaporte hacia Foto.
- Pasaporte ↔ Titular: Es una asociación bidireccional. Un Pasaporte está asociado a un Titular y un Titular está asociado a un Pasaporte. Ambas clases tienen una referencia la una a la otra.

En la clase Pasaporte, la composición con Foto se implementa en el constructor, donde se crea la instancia de Foto. La asociación bidireccional con Titular se establece en el constructor del pasaporte y, además, se actualiza la referencia en el objeto Titular (this.titular.setPasaporte(this)), para que ambas referencias se apunten mutuamente.

2. Celular - Batería - Usuario

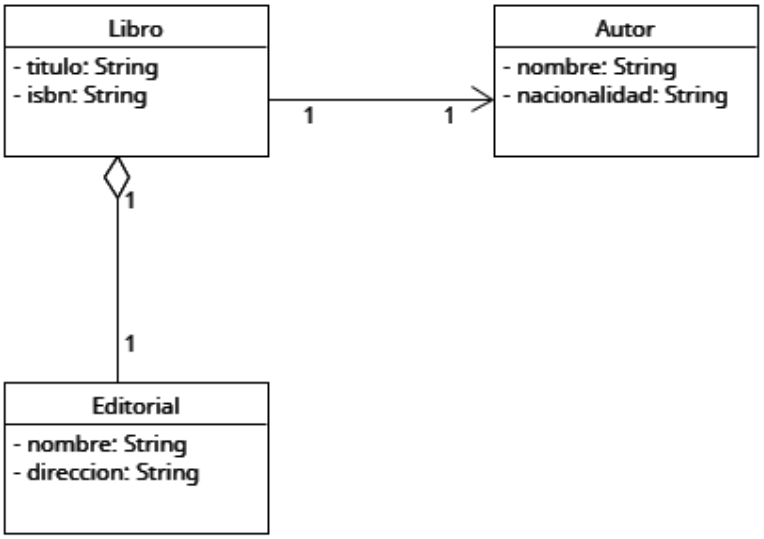


Relaciones:

- Celular → Batería: Esta es una relación de agregación. Un Celular "tiene" una Batería, pero la Batería puede existir por sí sola (por ejemplo, como un repuesto). Su ciclo de vida no depende del Celular. La dirección es unidireccional, del Celular a la Batería.
- Celular ↔ Usuario: Esta es una asociación bidireccional. Un Celular pertenece a un Usuario y un Usuario tiene un Celular. Ambas clases se conocen y tienen una referencia la una a la otra.

En la clase Celular, la agregación con Bateria se gestiona a través del constructor, donde se recibe el objeto Bateria ya instanciado, en lugar de crearlo internamente. La asociación bidireccional con Usuario se maneja de manera similar al ejercicio anterior, para que ambas clases tengan referencias mutuas.

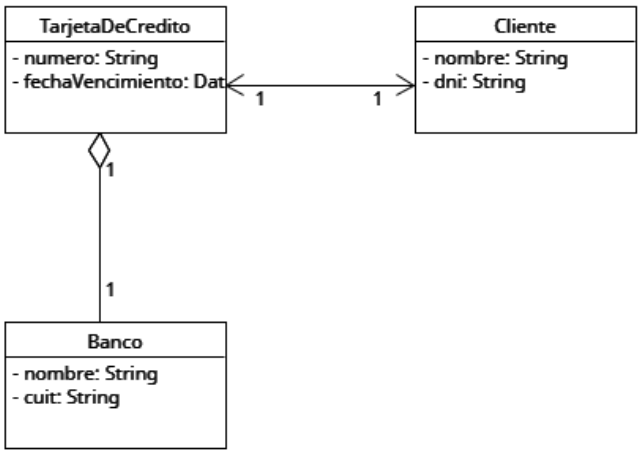
3. Libro - Autor - Editorial



Relaciones:

- Libro → Autor: Se trata de una asociación unidireccional. Un Libro conoce a su Autor, pero el Autor no tiene por qué conocer los libros que ha escrito. Es una relación "conoce a".
- Libro → Editorial: Esta es una relación de agregación. Un Libro "es publicado por" una Editorial. La Editorial puede existir de forma independiente sin necesidad de que el libro exista. Por ejemplo, una editorial puede cerrar, pero sus libros publicados siguen existiendo.

4. TarjetaDeCrédito - Cliente - Banco

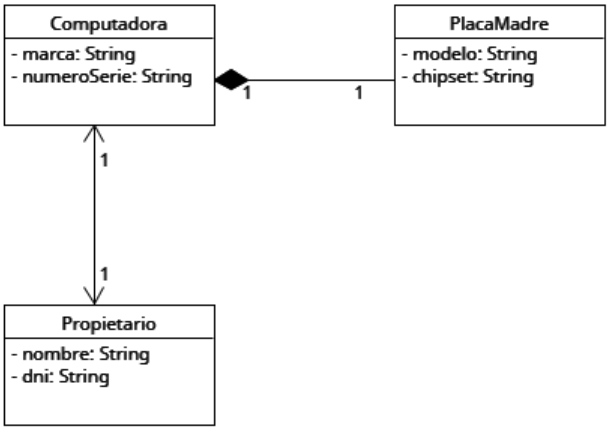


Relaciones:

- TarjetaDeCrédito ↔ Cliente: Se trata de una asociación bidireccional. Una TarjetaDeCrédito pertenece a un Cliente, y un Cliente tiene una TarjetaDeCrédito (en este caso, una sola para el modelo 1 a 1). Ambas clases tienen una referencia mutua.
- TarjetaDeCrédito → Banco: Esta es una relación de agregación. Una TarjetaDeCrédito es emitida por un Banco, pero el Banco es una entidad independiente que puede existir por sí misma, sin necesidad de que exista una tarjeta de crédito específica. Su ciclo de vida es independiente.

El código demuestra cómo la asociación bidireccional permite que tanto el Cliente como la TarjetaDeCrédito se refieran mutuamente, mientras que la agregación muestra que el Banco es un objeto independiente que es referenciado por la TarjetaDeCrédito.

5. Computadora - PlacaMadre - Propietario



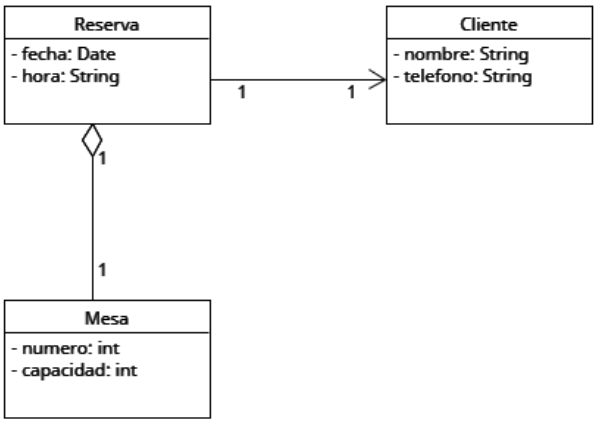
Relaciones:

- Computadora → PlacaMadre: Se trata de una relación de composición. La PlacaMadre es una parte fundamental y dependiente de la Computadora. No puede existir por sí misma fuera de una computadora. Su ciclo de vida está intrínsecamente ligado al de la computadora; si la Computadora se destruye, la PlacaMadre también lo hace.

- Computadora ↔ Propietario: Esta es una asociación bidireccional. Una Computadora pertenece a un Propietario, y un Propietario es el dueño de una Computadora. Ambas clases tienen una referencia mutua, lo que les permite conocerse entre sí.

En la clase Computadora, la composición con PlacaMadre se implementa creando la instancia de PlacaMadre directamente dentro del constructor. La asociación bidireccional con Propietario se maneja al pasar el objeto Propietario al constructor y, posteriormente, se establece la referencia inversa en el objeto Propietario.

6. Reserva - Cliente - Mesa

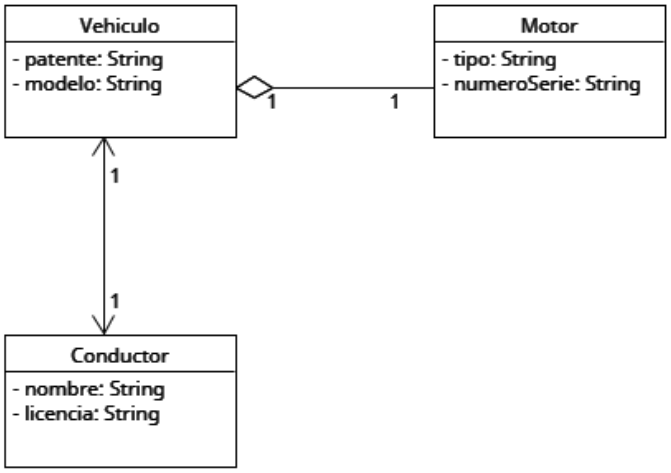


Relaciones:

- Reserva → Cliente: Se trata de una asociación unidireccional. La Reserva "pertenece a" un Cliente; el objeto Reserva conoce a su Cliente. Sin embargo, en este modelo, el Cliente no necesita tener una referencia a la Reserva.
- Reserva → Mesa: Esta es una relación de agregación. Una Reserva "ocupa" una Mesa, pero la Mesa es una entidad independiente que puede existir por sí sola en el restaurante, incluso cuando no tiene una reserva asignada. La Reserva se refiere a la Mesa, pero el ciclo de vida de la Mesa es independiente.

El código demuestra cómo la asociación unidireccional permite que la Reserva conozca a su Cliente sin que el Cliente conozca la Reserva. Por otro lado, la agregación se implementa al recibir un objeto Mesa ya creado en el constructor de la Reserva, lo que muestra su existencia independiente.

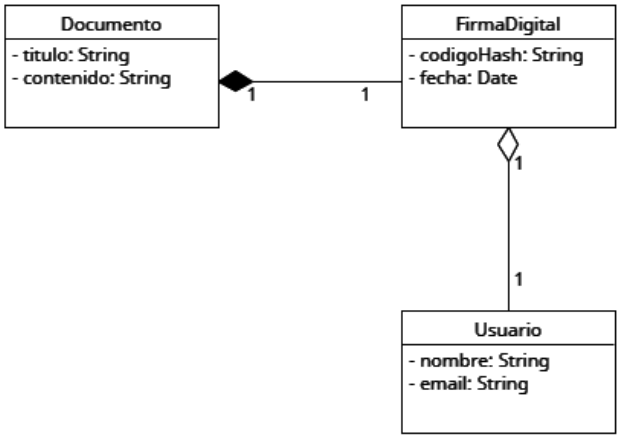
7. Vehículo - Motor - Conductor



Relaciones:

- Vehículo → Motor: Esta es una relación de agregación. Un Vehículo "usa" un Motor, pero el Motor puede existir de manera independiente (como un repuesto o en una tienda de autopartes). Su ciclo de vida no depende del vehículo.
- Vehículo ↔ Conductor: Se trata de una asociación bidireccional. Un Vehículo tiene un Conductor y un Conductor maneja un Vehículo. Ambas clases se conocen mutuamente y tienen referencias entre sí.

8. Documento - FirmaDigital - Usuario



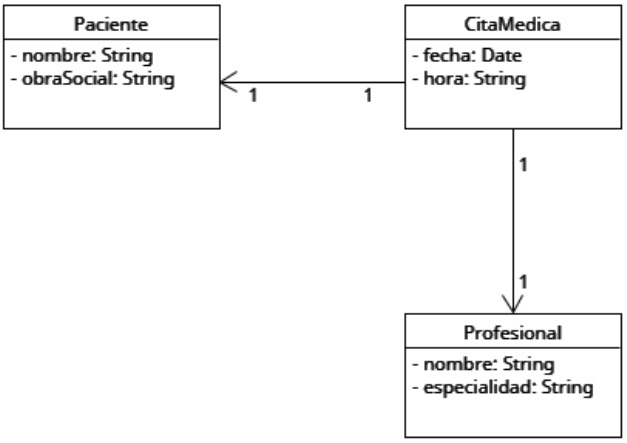
Relaciones:

- Documento → FirmaDigital: Se trata de una relación de composición. La FirmaDigital es una parte integral y dependiente de un Documento. La firma no tiene sentido ni existencia sin el documento al que está asociada. Cuando el Documento se destruye, la FirmaDigital asociada también se considera inválida o eliminada. La dirección es unidireccional.
- FirmaDigital → Usuario: Esta es una relación de agregación. Una FirmaDigital "pertenece a" un Usuario, pero el Usuario es una entidad independiente que puede existir sin necesidad de haber firmado un documento. El ciclo de vida del Usuario es independiente del de la

FirmaDigital. La FirmaDigital se refiere al Usuario para validar la autenticidad, pero el Usuario puede seguir existiendo y generar otras firmas. La dirección es unidireccional.

El código ilustra cómo la composición entre Documento y FirmaDigital se maneja al instanciar la firma dentro del constructor del documento. La agregación se implementa al pasar un objeto Usuario ya existente al constructor de la FirmaDigital, demostrando que el Usuario puede existir independientemente de la firma.

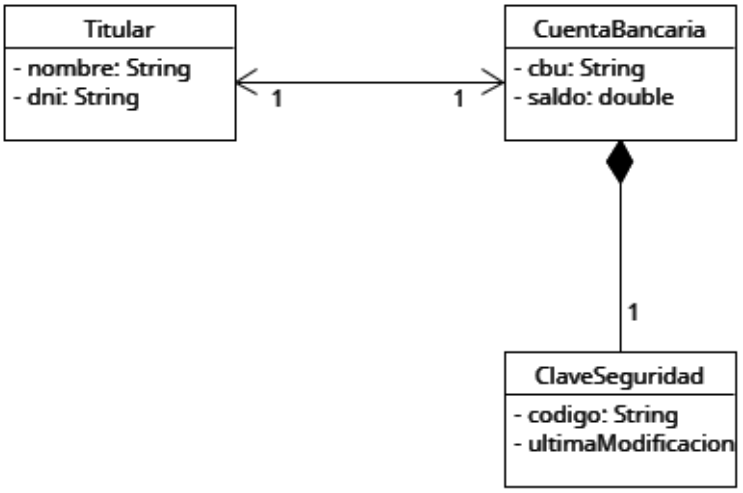
9. CitaMédica - Paciente - Profesional



Relaciones:

- CitaMédica → Paciente: Esta es una asociación unidireccional. La CitaMédica tiene una referencia al Paciente que asistirá a la consulta. Sin embargo, en este modelo, el Paciente no necesita una referencia de regreso a la cita.
- CitaMédica → Profesional: Similar a la relación anterior, es una asociación unidireccional. La CitaMédica tiene una referencia al Profesional que la atenderá. El Profesional no necesita una referencia a la cita en particular.

10. CuentaBancaria - ClaveSeguridad - Titular

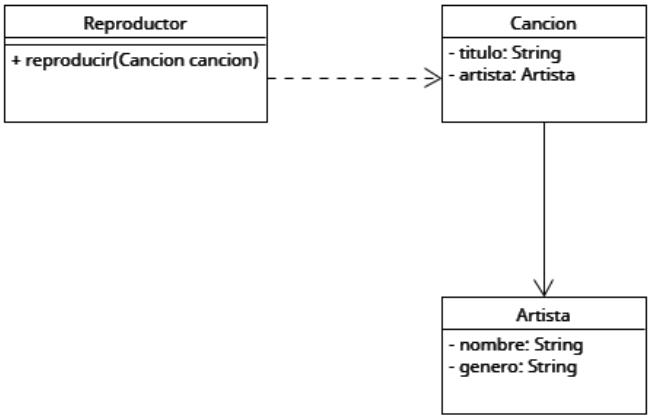


Relaciones:

- CuentaBancaria → ClaveSeguridad: Se trata de una relación de composición. La ClaveSeguridad es una parte intrínseca de la CuentaBancaria. No puede existir por sí sola sin una cuenta asociada. Si la cuenta se elimina, la clave de seguridad deja de ser relevante y se elimina con ella. La dirección es unidireccional.
- CuentaBancaria ↔ Titular: Esta es una asociación bidireccional. Una CuentaBancaria pertenece a un Titular y, a su vez, el Titular es dueño de la CuentaBancaria. Ambas clases tienen referencias mutuas entre sí.

Ejercicios de Dependencia de Uso

11. Reproductor - Canción - Artista



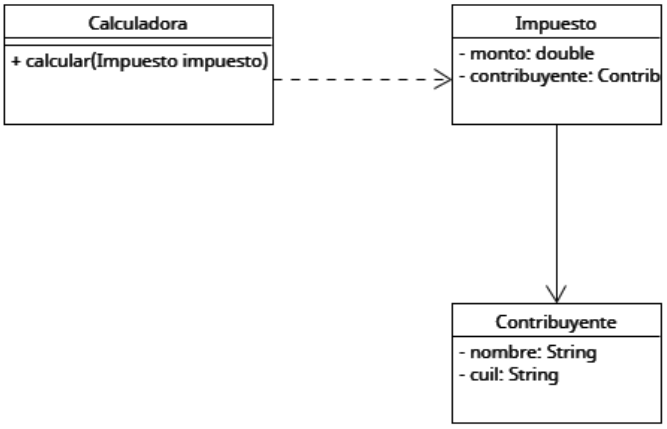
Relaciones:

- Canción → Artista: Se trata de una asociación unidireccional. Una Canción está asociada a un Artista, por lo que la clase Canción tiene una referencia (atributo) a la clase Artista. Sin embargo, la clase Artista no tiene una referencia de vuelta a la Canción en este modelo.

- Reproductor → Canción: Es una dependencia de uso. El Reproductor "usa" la clase Canción para su método reproducir(), pero no mantiene una referencia a Canción como un atributo de clase. La Canción se pasa como un parámetro en un método.

La dependencia de uso se evidencia en el método reproducir de la clase Reproductor, que recibe un objeto de la clase Cancion como parámetro para realizar su función. Esto significa que Reproductor depende de Cancion para operar, pero no la almacena como un atributo de clase.

12. Impuesto - Contribuyente - Calculadora



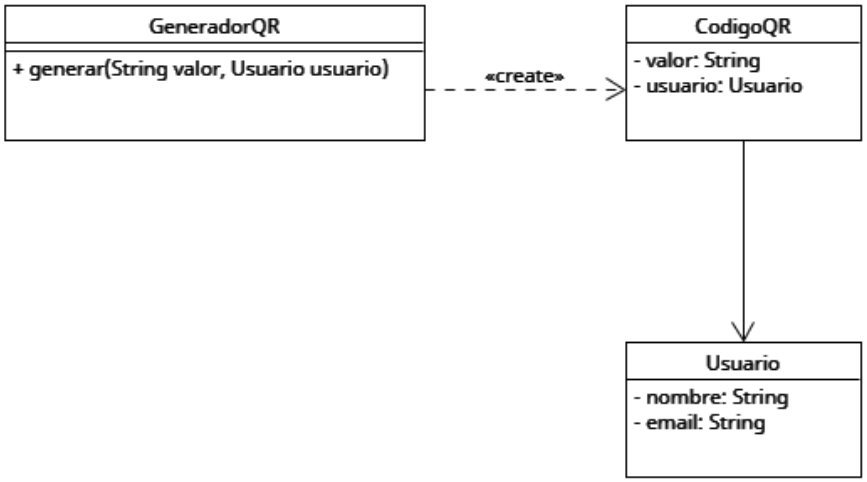
Relaciones:

- Impuesto → Contribuyente: Esta es una asociación unidireccional. La clase Impuesto tiene una referencia (atributo) al Contribuyente que debe pagarlo. Sin embargo, el Contribuyente no necesita tener una referencia al impuesto en este modelo.
- Calculadora → Impuesto: Se trata de una dependencia de uso. La clase Calculadora utiliza la clase Impuesto como un parámetro en su método calcular(), pero no la almacena como un atributo de clase. Esto significa que la Calculadora depende de la Impuesto para realizar su operación, pero no tiene una relación de "tiene un".

La dependencia de uso se hace evidente en el método calcular de la clase Calculadora, el cual recibe un objeto de la clase Impuesto como parámetro para realizar su operación, pero no lo retiene como un atributo de clase. Esto demuestra que Calculadora depende de la existencia de Impuesto para funcionar, pero no lo tiene como parte de su estructura.

Ejercicios de Dependencia de Creación

13. GeneradorQR - Usuario - CódigoQR

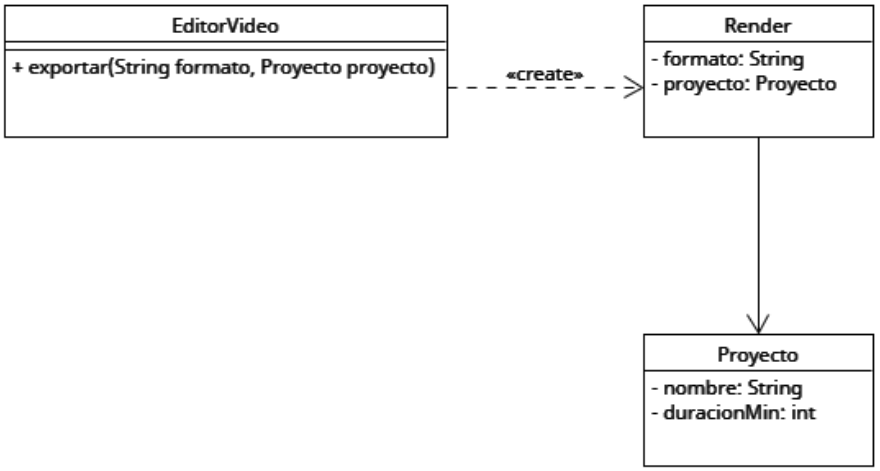


Relaciones:

- CódigoQR → Usuario: Se trata de una asociación unidireccional. Un CódigoQR está asociado a un Usuario, lo que significa que la clase CódigoQR tiene una referencia (atributo) a la clase Usuario. Sin embargo, el Usuario no tiene una referencia de vuelta al CódigoQR en este modelo.
- GeneradorQR → CódigoQR: Esta es una dependencia de creación. La clase GeneradorQR crea una instancia de CódigoQR dentro de su método generar(), pero no la almacena como un atributo. Esto demuestra que GeneradorQR depende de CódigoQR para realizar su función de creación, pero no la "posee" permanentemente.

En el método generar de la clase GeneradorQR, se crea (new) un nuevo objeto CodigoQR. Este objeto se devuelve, pero no se mantiene como un atributo de la clase GeneradorQR, lo que cumple con la definición de dependencia de creación.

14. EditorVideo - Proyecto - Render



Relaciones:

- Render → Proyecto: Es una asociación unidireccional. Un Render está vinculado a un Proyecto, por lo que la clase Render tiene una referencia (atributo) a la clase Proyecto. El Proyecto, en este modelo, no tiene una referencia de vuelta al Render.

- EditorVideo → Render: Esta es una dependencia de creación. El EditorVideo crea una instancia de Render dentro de su método exportar(). No la almacena como un atributo de clase. Esto significa que EditorVideo depende de Render para su operación de exportación, pero no la "posee" de forma permanente.
-

En el método exportar de la clase EditorVideo, se crea (new) un nuevo objeto Render y se lo devuelve. La clase EditorVideo no tiene un atributo Render, lo que demuestra que su relación es una dependencia de creación, ya que solo depende de Render para generar la instancia en tiempo de ejecución.