

PROGRAMACIÓN II

Trabajo Práctico 5: Relaciones UML 1 a 1

Alumno Santiago Raúl Salinas


Caso práctico

Ejercicio 1:


Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Pasaporte → Foto	Composición (Fuerte)	Unidireccional	La clase Pasaporte crea (new Foto(...)) la instancia de Foto dentro de su constructor, vinculando su ciclo de vida.
Pasaporte ↔ Titular	Asociación	Bidireccional	Ambas clases tienen una referencia a la otra. El método setTitular de Pasaporte debe sincronizar la referencia en la clase Titular (y viceversa, aunque aquí solo se implementa la sincronización en un sentido).

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Pasaporte	- numero: String	Pasaporte es el contenedor de Foto (Rombo Negro ) y conoce a Titular.
	- fechaEmision: String	
	- foto: Foto	
	- titular: Titular	
Foto	- imagen: String	Es la parte contenida.
	- formato: String	
Titular	- nombre: String	Es conocido por Pasaporte y conoce a Pasaporte.
	- dni: String	
	- pasaporte: Pasaporte	

Representación de Relaciones:


- **Composición (Pasaporte → Foto):** Una línea sólida con un **rombo negro** () en el lado de Pasaporte. Cardinalidad **1:1**.
- **Asociación Bidireccional (Pasaporte ↔ Titular):** Una línea sólida **sin flechas** (o con flechas en ambos lados) que conecta ambas clases. Cardinalidad **1:1**.

Ejercicio 2:

Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Celular → Bateria	Agregación (Medio)	Unidireccional	La clase Celular recibe un objeto Bateria ya creado como parámetro en su constructor y lo almacena como atributo. La Bateria puede existir sin el Celular.
Celular ↔ Usuario	Asociación	Bidireccional	Ambas clases tienen una referencia a la otra. El constructor/setter de Celular debe sincronizar la referencia en la clase Usuario.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Celular	- imei: String	Celular es el contenedor de Bateria (Rombo Blanco ) y conoce a Usuario.
	- marca: String	
	- modelo: String	
	- bateria: Bateria	
	- usuario: Usuario	
Bateria	- modelo: String	Es la parte, existe independientemente.
	- capacidad: String	
Usuario	- nombre: String	Es conocido por Celular y conoce a Celular.
	- dni: String	
	- celular: Celular	

Ejercicio 3:

Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Libro → Autor	Asociación (Débil)	Unidireccional	La clase Libro contiene un atributo de tipo Autor, pero Autor no tiene conocimiento del Libro. Se establece la referencia en el constructor de Libro.
Libro → Editorial	Agregación (Medio)	Unidireccional	La clase Libro recibe un objeto Editorial ya creado como parámetro. La Editorial es una parte que puede existir completamente independiente del Libro (Agregación).

Diagrama UML (Conceptual) :

Clase	Atributos	Visibilidad UML
Libro	- titulo: String	Libro conoce a Autor y a Editorial (Rombo Blanco <input type="checkbox"/>).
	- isbn: String	
	- autor: Autor	
	- editorial: Editorial	
Autor	- nombre: String	No conoce al Libro.
	- nacionalidad: String	
Editorial	- nombre: String	No conoce al Libro; existe independientemente.
	- direccion: String	

Representación de Relaciones:

- **Asociación Unidireccional (Libro → Autor):** Línea sólida con una **flecha abierta** en el lado de Autor. Cardinalidad **1:1**.
- **Agregación Unidireccional (Libro → Editorial):** Línea sólida con un **rombo blanco** (☐) en el lado de Libro y una flecha abierta en el lado de Editorial. Cardinalidad **1:1**.

Ejercicio 4:

Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
TarjetaDeCrédito ↔ Cliente	Asociación Bidireccional	Bidireccional	Ambas clases tienen una referencia a la otra. El constructor/setter de TarjetaDeCredito debe sincronizar la referencia en el objeto Cliente y viceversa.
TarjetaDeCrédito → Banco	Agregación	Unidireccional	TarjetaDeCredito recibe una referencia al objeto Banco ya creado como parámetro y lo almacena. El Banco puede existir sin la tarjeta.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
TarjetaDeCrédito	- numero: String	TarjetaDeCrédito contiene una referencia a Cliente y a Banco (Rombo Blanco <input type="checkbox"/>).
	- fechaVencimiento: String	
	- cliente: Cliente	

	- banco: Banco	
Cliente	- nombre: String	Es conocido por TarjetaDeCrédito y conoce a TarjetaDeCrédito.
	- dni: String	
	- tarjeta: TarjetaDeCredito	
Banco	- nombre: String	Es la parte en la Agregación (Rombo Blanco ◻), existe independientemente.
	- cuil: String	

El diagrama muestra las dos relaciones: la bidireccional (Cliente y TarjetaDeCrédito) y la agregación (TarjetaDeCrédito y Banco).

- **Asociación Bidireccional (TarjetaDeCrédito ↔ Cliente):** Línea sólida sin flechas (o con flechas en ambos lados).
- **Agregación Unidireccional (TarjetaDeCrédito → Banco):** Línea sólida con un **rombo blanco** (◻) en el lado de la clase contenedora (TarjetaDeCrédito) y una flecha abierta hacia Banco.

Ejercicio 5:

Análisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Computadora → PlacaMadre	Composición (Fuerte)	Unidireccional	La Computadora es dueña exclusiva de la PlacaMadre; la crea (new PlacaMadre(...)) dentro de su constructor, vinculando fuertemente el ciclo de vida.
Computadora ↔ Propietario	Asociación	Bidireccional	Ambas clases contienen una referencia a la otra. El constructor de Computadora debe asegurarse de que la referencia cruzada en Propietario se establezca correctamente (sincronización).

Diagrama UML (Conceptual)

Clase	Atributos	Visibilidad UML
Computadora	- marca: String	Contenedora de PlacaMadre (Rombo Negro ◼) y conoce a Propietario.
	- numeroSerie: String	
	- placaMadre: PlacaMadre	
	- propietario: Propietario	
PlacaMadre	- modelo: String	Es la parte contenida, no existe sin la Computadora.
	- chipset: String	
Propietario	- nombre: String	Es conocido por Computadora y conoce a Computadora.
	- dni: String	
	- computadora: Computadora	

El ejercicio combina una **Composición** y una **Asociación Bidireccional**. Cada una tiene una representación gráfica específica en UML:

Composición: Computadora → PlacaMadre (Relación Fuerte)

- **Representación:** Línea sólida con un **rombo negro (■)** relleno en el lado de la clase contenedora (**Computadora**).
- **Significado:** Esto indica que la PlacaMadre es una parte esencial de la Computadora y no puede existir si la Computadora es destruida (Composición). La dirección del vínculo es de la Computadora a la PlacaMadre.

Asociación Bidireccional: Computadora ↔ Propietario (Ambos se conocen)

- **Representación:** Línea sólida **sin flechas** que conecta ambas clases.
- **Significado:** Esto indica que la Computadora tiene una referencia al Propietario, y el Propietario tiene una referencia de vuelta a la Computadora, estableciendo una relación de conocimiento mutuo y permanente.

Ejercicio 6:

Análisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Reserva → Cliente	Asociación (Unidireccional)	Unidireccional	La clase Reserva almacena un atributo de tipo Cliente (referencia). El Cliente no conoce la Reserva.
Reserva → Mesa	Agregación	Unidireccional	La Reserva recibe una referencia a un objeto Mesa ya existente (parte). La Mesa es una entidad que existe por sí misma en el restaurante, independientemente de la Reserva.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Reserva	- fecha: String	La clase central. Contenedora de Mesa (□) y conoce a Cliente.
	- hora: String	
	- cliente: Cliente	
	- mesa: Mesa	
Cliente	- nombre: String	Es conocido por la Reserva, pero no conoce la Reserva.
	- telefono: String	
Mesa	- numero: int	Es la parte en la Agregación; existe en el restaurante independientemente.
	- capacidad: int	

- **Asociación Unidireccional (Reserva → Cliente):** Línea sólida con una **flecha abierta** en el lado de Cliente.
- **Agregación Unidireccional (Reserva → Mesa):** Línea sólida con un **rombo blanco** (◻) en el lado de la contenedora (Reserva) y una flecha abierta hacia Mesa.

Ejercicio 7:

Análisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Vehículo → Motor	Agregación	Unidireccional	El Vehículo recibe una referencia a un objeto Motor ya creado en su constructor. El motor puede venderse o existir en el stock sin un vehículo asignado.
Vehículo ↔ Conductor	Asociación	Bidireccional	Ambas clases tendrán una referencia a la otra. El constructor/setter de Vehículo se encargará de sincronizar la referencia cruzada en la clase Conductor.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Vehículo	- patente: String	Contenedor de Motor (◻) y conoce a Conductor.
	- modelo: String	
	- motor: Motor	
	- conductor: Conductor	
Motor	- tipo: String	Parte en la Agregación; existe por sí mismo.
	- numeroSerie: String	
Conductor	- nombre: String	Es conocido por Vehículo y conoce a Vehículo.
	- licencia: String	
	- vehiculo: Vehiculo	

- **Agregación (Vehículo → Motor):** Línea sólida con un **rombo blanco** (◻) en el lado de la clase contenedora (**Vehículo**).
- **Asociación Bidireccional (Vehículo ↔ Conductor):** Línea sólida **sin flechas** que conecta ambas clases.

Ejercicio 8:

Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Documento → FirmaDigital	Composición (Fuerte)	Unidireccional	El Documento es dueño exclusivo de la FirmaDigital. La FirmaDigital debe crearse dentro del constructor de Documento usando los datos primitivos de la firma, y no puede existir si el documento se destruye.
FirmaDigital → Usuario	Agregación (Medio)	Unidireccional	La FirmaDigital necesita una referencia al objeto Usuario (el firmante), pero el Usuario existe independientemente del documento o de la firma.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Documento	- titulo: String	Contenedor de FirmaDigital (Rombo Negro ■).
	- contenido: String	
	- firma: FirmaDigital	
FirmaDigital	- codigoHash: String	Parte de la Composición, pero a su vez, Contenedora del Usuario (Rombo Blanco □).
	- fecha: String	
	- usuario: Usuario	
Usuario	- nombre: String	Parte de la Agregación; existe por sí mismo.
	- email: String	

- **Composición (Documento → FirmaDigital):** Línea sólida con un **rombo negro** (■) relleno en el lado de la clase contenedora (**Documento**).
- **Agregación (FirmaDigital → Usuario):** Línea sólida con un **rombo blanco** (□) en el lado de la clase contenedora (**FirmaDigital**) y una flecha abierta hacia Usuario.

Ejercicio 9:

Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
CitaMédica → Paciente	Asociación (Unidireccional)	Unidireccional	La clase CitaMédica almacena un atributo de tipo Paciente (referencia) recibido en el constructor. El Paciente no conoce la CitaMédica.
CitaMédica → Profesional	Asociación (Unidireccional)	Unidireccional	La clase CitaMédica almacena un atributo de tipo Profesional (referencia) recibido en el constructor. El Profesional no conoce la CitaMédica.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
CitaMédica	- fecha: String	Conoce a Paciente y a Profesional.
	- hora: String	
	- paciente: Paciente	
	- profesional: Profesional	
Paciente	- nombre: String	Es conocido por CitaMédica.
	- obraSocial: String	
Profesional	- nombre: String	Es conocido por CitaMédica.
	- especialidad: String	

- **Asociación Unidireccional (CitaMédica → Paciente):** Línea sólida con una **flecha abierta** en el lado de Paciente.
- **Asociación Unidireccional (CitaMédica → Profesional):** Línea sólida con una **flecha abierta** en el lado de Profesional.

Ejercicio 10:

Este es el décimo y último ejercicio, que requiere implementar una **Composición** para la relación fuerte (CuentaBancaria y ClaveSeguridad) y una **Asociación Bidireccional** para el conocimiento mutuo (CuentaBancaria y Titular).

Análisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
CuentaBancaria → ClaveSeguridad	Composición (Fuerte)	Unidireccional	La CuentaBancaria crea la ClaveSeguridad dentro de su constructor (new ClaveSeguridad(...)), vinculando sus ciclos de vida.
CuentaBancaria ↔ Titular	Asociación	Bidireccional	Ambas clases contienen una referencia a la otra. El constructor/setter de CuentaBancaria debe sincronizar la referencia cruzada en Titular.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
CuentaBancaria	- cbu: String	Contenedora de ClaveSeguridad (■) y conoce a Titular.
	- saldo: double	
	- clave: ClaveSeguridad	
	- titular: Titular	
ClaveSeguridad	- codigo: String	Es la parte, no puede existir sin la CuentaBancaria.
	- ultimaModificacion: String	
Titular	- nombre: String	Es conocido por CuentaBancaria y conoce a CuentaBancaria.

	- dni: String	
	- cuenta: CuentaBancaria	

- **Composición (CuentaBancaria → ClaveSeguridad):** Línea sólida con un **rombo negro** (■) relleno en el lado de la clase contenedora (CuentaBancaria).
- **Asociación Bidireccional (CuentaBancaria ↔ Titular):** Línea sólida **sin flechas** que conecta ambas clases.

DEPENDENCIA DE USO

Ejercicio 11:

Este es un ejercicio que implementa la **Dependencia de Uso**, una relación temporal donde una clase solo utiliza los servicios de otra a través de un parámetro de método, sin guardarla como atributo.

Análisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Canción → Artista	Asociación (Unidireccional)	Unidireccional	La clase Canción almacena una referencia al objeto Artista recibido en el constructor. El Artista no conoce la Canción.
Reproductor.reproducir(Cancion)	Dependencia de Uso	Unidireccional	El método reproducir de Reproductor acepta un objeto Canción como parámetro. Reproductor no tiene un atributo Canción, por lo que el vínculo es puramente temporal mientras el método se ejecuta.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Canción	- titulo: String	Contiene una referencia a Artista.
	- artista: Artista	
Artista	- nombre: String	Es conocido por Canción.
	- genero: String	
Reproductor	(Sin atributos relacionados)	Depende de Canción para el método reproducir.
	+ reproducir(cancion: Cancion): void	

- **Asociación Unidireccional (Canción → Artista):** Línea sólida con una **flecha abierta** apuntando a Artista.

- **Dependencia de Uso (Reproductor → Canción):** Línea **discontinua o punteada** con una **flecha abierta** apuntando a Canción. Generalmente se usa la etiqueta <<use>> o <<call>> sobre la línea para indicar el tipo de dependencia.

Ejercicio 12:

Este ejercicio requiere implementar una **Asociación Unidireccional** (Impuesto conoce al Contribuyente) y una **Dependencia de Uso** (Calculadora usa el objeto Impuesto temporalmente en un método).

Análisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Impuesto → Contribuyente	Asociación (Unidireccional)	Unidireccional	La clase Impuesto almacena la referencia al objeto Contribuyente (a quién pertenece el impuesto) como un atributo permanente. El Contribuyente no tiene conocimiento del Impuesto.
Calculadora.calcular(Impuesto)	Dependencia de Uso	Unidireccional	La clase Calculadora utiliza el objeto Impuesto únicamente como parámetro de un método. El objeto Impuesto no se guarda como atributo en Calculadora, por lo que la relación es temporal.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Impuesto	- monto: double	Contiene una referencia a Contribuyente.
	- contribuyente: Contribuyente	
Contribuyente	- nombre: String	Es conocido por Impuesto.
	- cuil: String	
Calculadora	(Sin atributos relacionados)	Depende de Impuesto para el método calcular.
	+ calcular(impuesto: Impuesto): void	

- **Asociación Unidireccional (Impuesto → Contribuyente):** Línea sólida con una **flecha abierta** apuntando a Contribuyente.

Dependencia de Uso (Calculadora → Impuesto): Línea **discontinua** con una **flecha abierta** apuntando a Impuesto, a menudo etiquetada con <<use>>.

DEPENDENCIA DE CREACIÓN

Ejercicio 13:

Este ejercicio introduce la **Dependencia de Creación**, donde una clase asume temporalmente la

responsabilidad de instanciar otra clase dentro de un método.

Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
CódigoQR → Usuario	Asociación (Unidireccional)	Unidireccional	La clase CódigoQR almacena una referencia permanente al objeto Usuario (quien posee el código), recibido en su constructor.
GeneradorQR.generar(..)	Dependencia de Creación	Unidireccional	La clase GeneradorQR utiliza la palabra clave new para crear una instancia de CódigoQR dentro del método generar. El objeto creado es local al método y no se guarda como atributo en GeneradorQR.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
CódigoQR	- valor: String	Contiene una referencia a Usuario.
	- usuario: Usuario	
Usuario	- nombre: String	Es conocido por CódigoQR.
	- email: String	
GeneradorQR	(Sin atributos relacionados)	Depende de CódigoQR para crearlo.
	+ generar(valor: String, usuario: Usuario): void	

- **Asociación Unidireccional (CódigoQR → Usuario):** Línea sólida con una **flecha abierta** apuntando a Usuario.
- **Dependencia de Creación (GeneradorQR → CódigoQR):** Línea **discontinua** con una **flecha abierta** apuntando a CódigoQR, etiquetada con <<create>>.

Ejercicio 14:

Este ejercicio combina la **Asociación Unidireccional** (Render necesita saber de qué Proyecto viene) con la **Dependencia de Creación** (EditorVideo genera el objeto Render).

Analisis de relaciones:

Relación	Tipo	Dirección	Implementación Clave en Java
Render → Proyecto	Asociación (Unidireccional)	Unidireccional	La clase Render almacena una referencia permanente al objeto Proyecto (el origen del renderizado), recibido en su constructor.
EditorVideo.exportar(...)	Dependencia de Creación	Unidireccional	La clase EditorVideo crea una instancia de Render dentro del método exportar usando el operador new. El Render no se guarda como atributo en EditorVideo, sino que es una responsabilidad temporal de creación.

Diagrama UML (Conceptual):

Clase	Atributos	Visibilidad UML
Render	- formato: String	Contiene una referencia a Proyecto.
	- proyecto: Proyecto	
Proyecto	- nombre: String	Es conocido por Render.
	- duracionMin: double	
EditorVideo	(Sin atributos relacionados)	Depende de Render para crearlo.
	+ exportar(formato: String, proyecto: Proyecto): void	

- **Asociación Unidireccional (Render → Proyecto):** Línea sólida con una **flecha abierta** apuntando a Proyecto.
- **Dependencia de Creación (EditorVideo → Render):** Línea **discontinua** con una **flecha abierta** apuntando a Render, etiquetada con <<create>>.