

**Trabajo Práctico N° 5**

**Relaciones UML 1 a 1**

Comision 16

Marinoni Macarena <[marinonimacarena@gmail.com](mailto:marinonimacarena@gmail.com)>

**Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional**

**Programación II**

**Profesor:** Cortez, Alberto

**Tutor:** Bianchi, Neyén

16/11/2025

## ÍNDICE

Objetivo general.....	3
Marco teórico.....	3
Ejercicios de Relaciones 1 a 1.....	4
Ejercicios de Dependencia de Uso.....	14
Ejercicios de Dependencia de Creación.....	16
Conclusión.....	17
Link a Repositorio GitHub:.....	18

## Objetivo general

Modelar clases con relaciones 1 a 1 utilizando diagramas UML. Identificar correctamente el tipo de relación (asociación, agregación, composición, dependencia) y su dirección, y llevarlas a implementación en Java.

## Marco teórico

Concepto	Aplicación en el proyecto
<b>Asociación</b>	Relación entre clases con referencia mutua o directa, puede ser uni o bidireccional
<b>Agregación</b>	Relación de "tiene un" donde los objetos pueden vivir independientemente
<b>Composición</b>	Relación fuerte de contención, el ciclo de vida del objeto contenido depende del otro
<b>Dependencia de uso</b>	Una clase usa otra como parámetro en un método, sin almacenarla como atributo
<b>Dependencia de creación</b>	Una clase crea otra en tiempo de ejecución, sin mantenerla como atributo

## 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

Objetivo: Modelar un pasaporte que incluye una foto propia y está asociado a un titular. El pasaporte crea y administra su foto (parte), mientras que el titular es una entidad independiente que conoce su pasaporte.

Diagrama UML

- Titular 1 <—————> 1 Pasaporte ◆————— 1 Foto  
(asociación bidi) (composición)

Tipo de relación y dirección

- Pasaporte–Foto: Composición (1:1). Unidireccional (el todo conoce a la parte)
- Pasaporte–Titular: Asociación (1:1). Bidireccional (ambos se conocen).

Implementación

```

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Resolucion-TP5 - Apache NetBeans I... Search (Ctrl+I)
Projects X Files Services
Main.java [-A] X
Source History ...
11  * @author vallett
12  /*
13   * Clase de prueba del Ejercicio 1
14  */
15 public class Main {
16     /**
17      * @param args the command line arguments
18     */
19     public static void main(String[] args) {
20         Titular t = new Titular("Marinoni, Macarena", "38.374.424");
21         Pasaporte p = new Pasaporte(
22             "AR-2025-0001",
23             LocalDate.of(2025, 3, 1),
24             t,
25             "/fotos/pasaporte-maca.jpg",
26             "JPEG"
27         );
28         System.out.println("==== Ejercicio 1 ====");
29         System.out.println(p);
30         System.out.println("Desde titular: " + t.getPasaporte());
31     }
32 }

```

```

Foto.java [-A] X Titular.java [-A] X Pasaporte.java [-A] X
Source History ...
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5 package resolucion.tp5.Ejercicio1;

```

```

Output X Javadoc
TUPaD_P2-C22025 - C:\Users\marin\Documents\GitHub\TUPaD_P2-C22025 X Resolucion-TP5 (run) X
run:
==== Ejercicio 1 ====
Pasaporte(numero='AR-2025-0001', fechaEmision=2025-03-01, titular=Titular[nombre='Macarena Marinoni', dni='38.374.424'], foto=Foto[formato='JPEG'])
Desde titular: Pasaporte(numero='AR-2025-0001', fechaEmision=2025-03-01, titular=Titular[nombre='Macarena Marinoni', dni='38.374.424'])
BUILD SUCCESSFUL (total time: 0 seconds)

```

## 2. Celular - Batería - Usuario

- a. Agregación: Celular → Batería
  - b. Asociación bidireccional: Celular ↔ Usuario

## Clases y atributos:

- i. Celular: imei, marca, modelo
  - ii. Batería: modelo, capacidad
  - iii. Usuario: nombre, dni

Objetivo: Representar un celular que usa una batería intercambiable y está asignado a un usuario. La batería no depende del ciclo de vida del celular.

## Diagrama UML

- Usuario 1 <----> 1 Celular o— 1 Bateria  
(asociación bidi) (agregación)

## Tipo de relación y dirección

- Celular–Batería: Agregación (1:1). Unidireccional (el celular conoce su batería).
  - Celular–Usuario: Asociación (1:1). Bidireccional (el usuario conoce su celular y viceversa).

## Implementación

The screenshot shows the Apache NetBeans IDE 26 interface. The top menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help, and a search bar. The title bar reads "Resolucion-TP5 - Apache NetBeans IDE 26".

The left sidebar displays the project structure under "Projects X". It lists numerous Java files (e.g., Bateria.java, Celular.java, Main.java, Usuario.java) and several packages (e.g., 00\_Concatenacion [main], 00\_Entradas [main], 00\_HolaMundo [main], etc.). A specific package, "Resolucion-TP5 (main)", is selected.

The main workspace shows the code editor for the "Main.java" file. The code defines a class "Main" with a static void main method that creates instances of "Bateria" and "Celular" and prints their details to the console.

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Clases/Class.java to edit this template
 */
package resolucion.tp5.Ejercicio2;

/**
 *
 * @author marinon
 */
public class Main {

    public static void main(String[] args) {
        Usuario u = new Usuario("Luis Gómez", "39.777.888");
        Bateria b = new Bateria("Li-Ion XZ-40", 5000);
        Celular c = new Celular("356789012345678", "Samsung", "A54", b, u);

        System.out.println("==> Ejercicio 2 ==>");
        System.out.println(c);
    }
}
```

The bottom right corner shows the output window with the command "run:" followed by the execution results and a "BUILD SUCCESSFUL" message.

### 3. Libro - Autor - Editorial

- a. Asociación unidireccional: Libro → Autor
- b. Agregación: Libro → Editorial

Clases y atributos:

- i. Libro: título, isbn
- ii. Autor: nombre, nacionalidad
- iii. Editorial: nombre, dirección

Objetivo Modelar un libro que conoce a su autor y su editorial. El autor y la editorial existen independientemente del libro.

Diagrama UML

- Libro —→ Autor
- Libro o—→ Editorial

Tipo de relación y dirección

- Libro-Autor: Asociación (1:1). Unidireccional (el libro conoce al autor).
- Libro-Editorial: Agregación (1:1). Unidireccional (el libro conoce a la editorial).

Implementación

```

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Reso... Search (Ctrl+F) 283.6 / 648.0 MB
Projects X Files Services <default config>
Autor.java [-/A] Editorial.java [-/A] Libro.java [-/A] Main.java [-/A]
Source History
1 /**
2  * Click nbfs://nbshost/SystemFileSystem/Templates/Licenses/license-default
3  * Click nbfs://nbshost/SystemFileSystem/Templates/Classes/Class.java to ed
4  */
5 package resolucion.tp5.Ejercicio3;
6
7 /**
8 *
9 * @author marinon
10 */
11 public class Main {
12
13     public static void main(String[] args) {
14         Autor autor = new Autor("Julio Cortazar", "Argentina");
15         Editorial ed = new Editorial("Alfaguara", "30-12345678-9");
16         Libro libro = new Libro("Rayuela", 1963, autor, ed);
17
18         System.out.println("== Ejercicio 3 ==");
19         System.out.println(libro);
20     }
21 }

```

Output X Javadoc

```

TUPaD_P2-C22025 - C:\Users\marin\Documents\GitHub\TUPaD_P2-C22025 Resolucion-TP5 (run)
run:
== Ejercicio 3 ==
Libro(titulo='Rayuela', anio=1963, autor=Autor(nombre='Julio Cortazar', nacionalidad='Argentina'))
BUILD SUCCESSFUL (total time: 0 seconds)

```

Finished building Resolucion-TP5 (run).

#### 4. TarjetaDeCrédito - Cliente - Banco

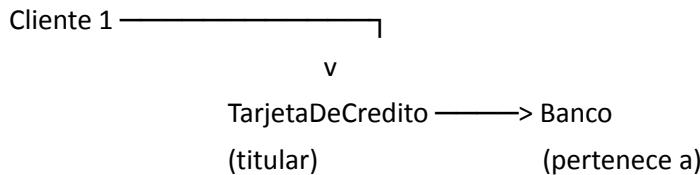
- a. Asociación bidireccional: TarjetaDeCrédito ↔ Cliente
- b. Agregación: TarjetaDeCrédito → Banco

Clases y atributos:

- i. TarjetaDeCrédito: numero, fechaVencimiento
- ii. Cliente: nombre, dni
- iii. Banco: nombre, cuit

Objetivo: Representar una tarjeta con número y fecha de vencimiento que pertenece a un banco y está asociada a un cliente titular.

Diagrama UML



Tipo de relación y dirección

- TarjetaDeCredito–Cliente: Asociación (1:1), con referencia inversa opcional en Cliente. Bidireccional (recomendado) o uni si no se requiere inversa.
- TarjetaDeCredito–Banco: Asociación (1:1). Unidireccional (la tarjeta conoce al banco).

Implementación

The screenshot shows an IDE interface with several Java files open in tabs. The main file is Main.java, which contains a main method that creates a Banco, a Cliente, and a TarjetaDeCredito, then prints their details. The output window shows the execution results.

```

public static void main(String[] args) {
    System.out.println("---- Ejercicio 4 -- Tarjeta, Cliente y Banco ----");
    // Creamos el banco
    Banco bancol = new Banco("Banco Nación", "30-12345678-9");

    // Creamos el cliente
    Cliente clientel = new Cliente("Maria Lopez", "40123456");

    // Creamos la tarjeta asociando cliente y banco
    TarjetaDeCredito tarjetal = new TarjetaDeCredito(
        "1234-5678-9012-3456",
        "12/28",
        clientel,
        bancol
    );

    // Mostramos resultados
    System.out.println(tarjetal);
    System.out.println("Cliente accede a su tarjeta: " + clientel.getTarjeta());
}
  
```

## 5. Computadora - PlacaMadre - Propietario

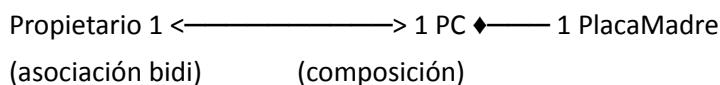
- a. Composición: Computadora → PlacaMadre
- b. Asociación bidireccional: Computadora ↔ Propietario

Clases y atributos:

- i. Computadora: marca, numeroSerie
- ii. PlacaMadre: modelo, chipset
- iii. Propietario: nombre, dni

Objetivo: Modelar una PC que contiene una placa madre (parte esencial) y está asociada a un propietario. La PC crea y administra su placa madre (composición). El propietario existe de forma independiente.

Diagrama UML



Tipo de relación y dirección

- PC-PlacaMadre: Composición (1:1). Unidireccional (la PC conoce su placa).
- PC-Propietario: Asociación (1:1). Bidireccional (opcional; aquí la implementamos).

Implementación

The screenshot shows an IDE interface with several tabs open. The main tab displays the following Java code:

```

package resolucion.tp5.Ejercicio5;

public class Main {
    public static void main(String[] args) {
        System.out.println("== Ejercicio 5: Computadora / PlacaMadre / Propietario ==");
        Propietario p = new Propietario("Sofia Diaz", "40.888.999");
        Computadora computadora = new Computadora("Lenovo", "ThinkStation", "Z790", "LGA1700", p);
        System.out.println(computadora);
        System.out.println("Desde propietario veo su Computadora: " + p.getPC());
    }
}
  
```

The code defines a `Main` class with a `main` method. It creates a `Propietario` object named `p` with name "Sofia Diaz" and dni "40.888.999". It then creates a `Computadora` object named `computadora` with brand "Lenovo", model "ThinkStation", chipset "Z790", and socket "LGA1700". Finally, it prints the computer details and the owner's view of the computer.

The left sidebar shows a project structure with multiple source packages and test packages. The output window at the bottom shows the execution results and a successful build message.

## 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: número, capacidad

Objetivo: Modelar una reserva de restaurante que usa una mesa existente y está asociada a un cliente. Ni el cliente ni la mesa dependen del ciclo de vida de la reserva.

Diagrama UML

Reserva o——> Mesa

Reserva —> Cliente

(agregación) (asociación)

Tipo de relación y dirección

- Reserva–Mesa: Agregación (1:1) — la mesa se inyecta; existe fuera.
- Reserva–Cliente: Asociación (1:1).
- Ambas unidireccionales desde Reserva (la reserva conoce a mesa y cliente).

Implementación

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5  package resolucion.tp5.Ejercicio6;
6
7  import java.time.LocalDateTime;
8
9  /**
10  *
11  * @author marinon
12  */
13 public class Main {
14     public static void main(String[] args) {
15         System.out.println("==== Ejercicio 6 — Reserva / Cliente / Mesa ====");
16
17         Mesa m = new Mesa(12, 4);
18         Cliente c = new Cliente("Martin Lopez", "+54 9 11 5555-5555");
19         Reserva r = new Reserva(LocalDateTime.of(2025, 10, 20, 20, 30),
20                                4, m, c);
21
22         System.out.println(r);
23     }
24 }

```

Output X Javadoc

TUPaD\_P2-C22025 - C:\Users\marin\Documents\GitHub\TUPaD\_P2-C22025 × Resolucion-TP5 (run) ×

```

run:
==== Ejercicio 6 ♦ Reserva / Cliente / Mesa ====
Reserva[fechaHora=2025-10-20T20:30, cantidadPersonas=4, mesa=Mesa[numero=12, capacidad=4], cliente=Cliente
BUILD SUCCESSFUL (total time: 0 seconds)

```

## 7. Vehículo - Motor - Conductor

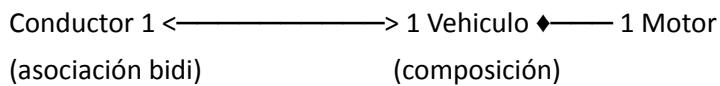
- a. Agregación: Vehículo → Motor
  - b. Asociación bidireccional: Vehículo ↔ Conductor

## Clases y atributos:

- i. Vehículo: patente, modelo
  - ii. Motor: tipo, numeroSerie
  - iii. Conductor: nombre, licencia

Objetivo: Modelar un vehículo que contiene su motor (parte esencial) y está asignado a un conductor. El vehículo crea su motor (composición). El conductor existe por separado y conoce su vehículo (asociación bidireccional).

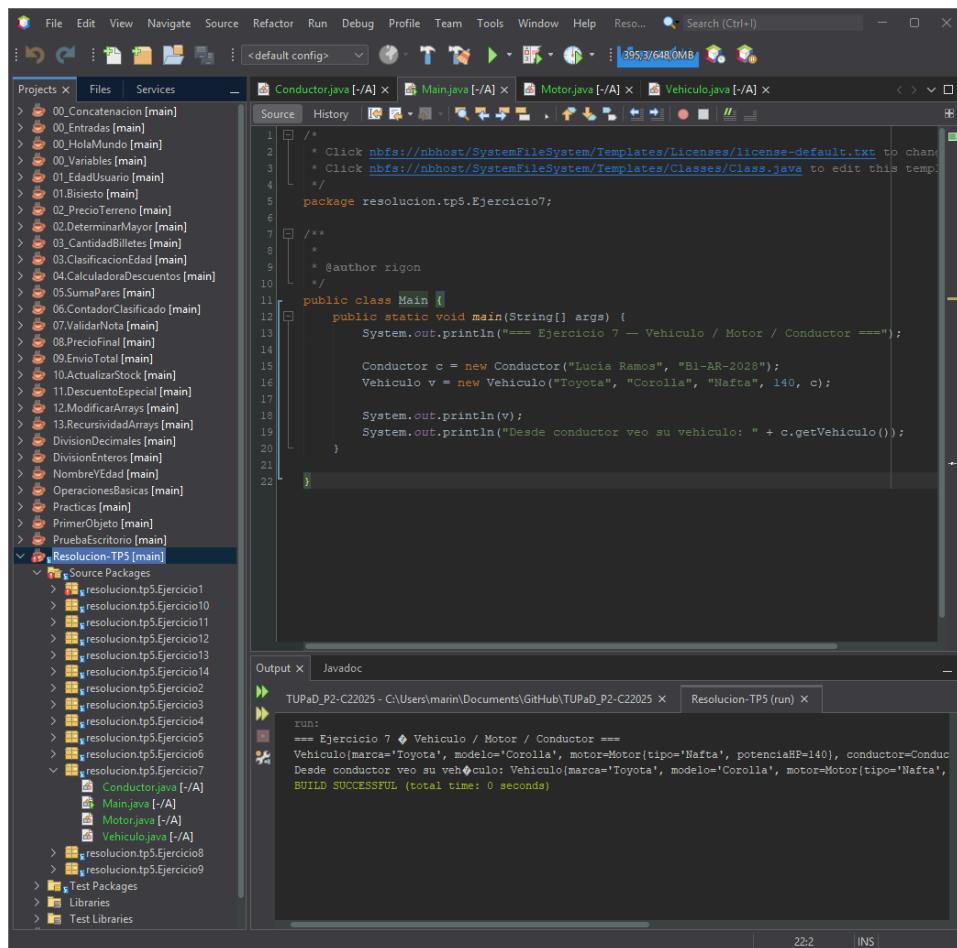
## Diagrama UML



## Relaciones y dirección

- Vehiculo–Motor: Composición (1:1), unidireccional desde Vehiculo.
  - Vehiculo–Conductor: Asociación (1:1), bidireccional.

## Implementación:



## 8. Documento - FirmaDigital - Usuario

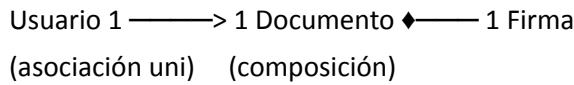
- a. Composición: Documento → FirmaDigital
- b. Agregación: FirmaDigital → Usuario

Clases y atributos:

- i. Documento: titulo, contenido
- ii. FirmaDigital: codigoHash, fecha
- iii. Usuario: nombre, email

Objetivo: Modelar un documento que incluye una firma única (parte) y está asociado a un usuario autor. El documento crea su firma (composición) a partir de datos crudos.

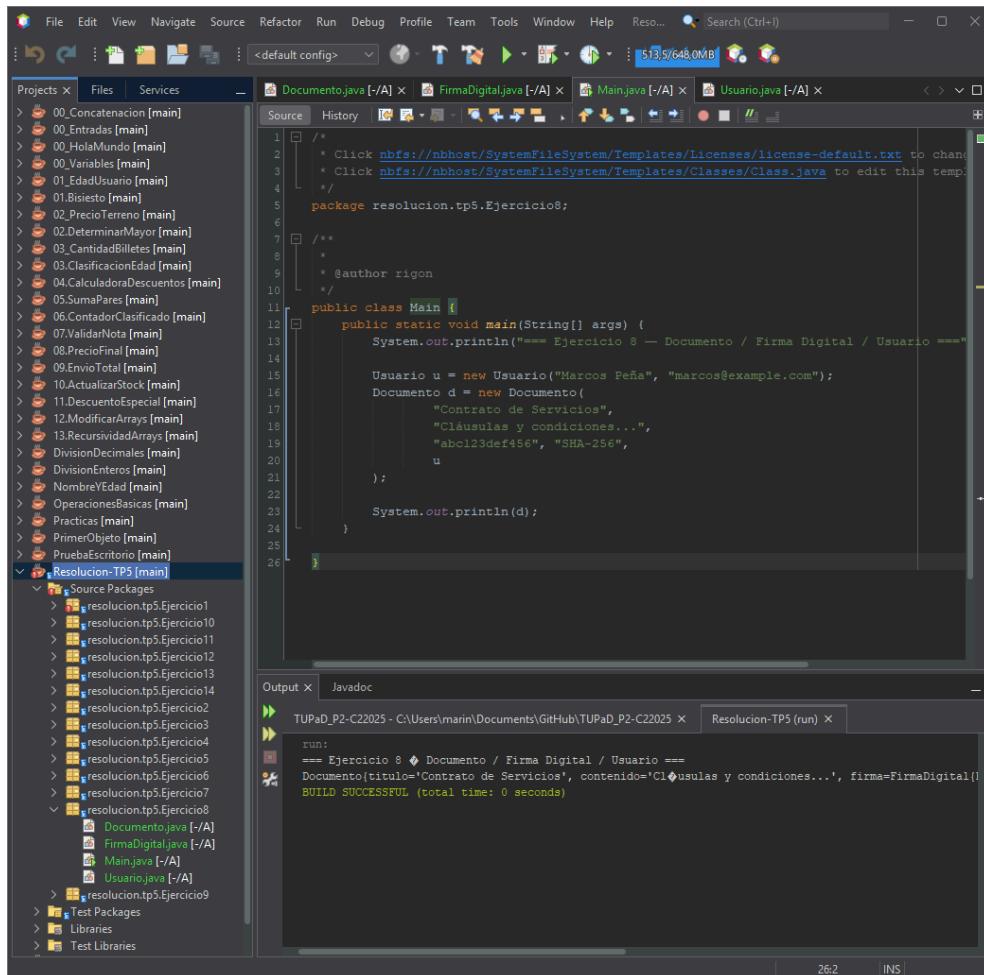
Diagrama UML



Relaciones y dirección

- Documento–Firma: Composición (1:1), unidireccional desde Documento.
- Documento–Usuario: Asociación (1:1), unidireccional desde Documento.

Implementación



The screenshot shows an IDE interface with several tabs open. The tabs include 'Documento.java [-A]', 'FirmaDigital.java [-A]', 'Main.java [-A]', and 'Usuario.java [-A]'. The 'Main.java' tab is active, displaying the following code:

```

1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license.
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template.
4  */
5  package resolucion.tp5.Ejercicio8;
6
7  /**
8  *
9  * @author rigon
10 */
11 public class Main {
12     public static void main(String[] args) {
13         System.out.println("== Ejercicio 8 == Documento / Firma Digital / Usuario ==");
14
15         Usuario u = new Usuario("Marcos Peña", "marcos@example.com");
16         Documento d = new Documento(
17             "Contrato de Servicios",
18             "Clausulas y condiciones...",
19             "abc123def456",
20             "SHA-256",
21             u);
22
23         System.out.println(d);
24     }
25 }
  
```

The 'Output' tab shows the results of a build:

```

TUPaD_P2-C22025 - C:\Users\marin\Documents\GitHub\TUPaD_P2-C22025 x Resolucion-TP5 (run) x
run:
--- Ejercicio 8 ♦ Documento / Firma Digital / Usuario ---
Documento[titulo='Contrato de Servicios', contenido='Clausulas y condiciones...', firma=FirmaDigital()]
BUILD SUCCESSFUL (total time: 0 seconds)
  
```

## 9. CitaMédica - Paciente - Profesional

- a. Asociación unidireccional: CitaMédica → Paciente,
  - b. Asociación unidireccional: CitaMédica → Profesional

## Clases y atributos:

- i. CitaMédica: fecha, hora
  - ii. Paciente: nombre, obraSocial
  - iii. Profesional: nombre, especialidad

Objetivo: Modelar una cita médica que usa a un paciente y a un profesional existentes (inyectados). Ninguno depende de la vida de la cita; la cita solo agrega/relaciona a ambos.

## Diagrama UML

Cita —> Paciente

## Cita —> Profesional

(asociación uni) (asociación uni)

## Relaciones y dirección

- Cita–Paciente: Asociación (1:1), unidireccional desde Cita.
  - Cita–Profesional: Asociación (1:1), unidireccional desde Cita.

## Implementación:

The screenshot shows the NetBeans IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help, Reso..., Search (Ctrl+I).
- Toolbar:** Standard NetBeans icons for file operations.
- Project Explorer:** Shows a project named "Resolucion-TP5 [main]" containing various Java files (e.g., 00\_Concatenacion, 00\_Entradas, 00\_HolaMundo, etc.) and source packages for "resolucion.tp5.Ejercicio1" through "resolucion.tp5.Ejercicio9".
- Code Editor:** The main editor window displays the "Main.java" file with the following code:

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license.
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template.
4   */
5  package resolucion.tp5.Ejercicio9;
6
7  import java.time.LocalDateTime;
8
9  /**
10  *
11  * @author marinon
12  */
13 public class Main {
14     public static void main(String[] args) {
15         System.out.println("==> Ejercicio 9 -- Cita / Paciente / Profesional ==>");
16
17         Paciente p = new Paciente("Andrea Ruiz", "40.222.333");
18         Profesional dr = new Profesional("Dr. Gómez", "MP 12345");
19         Cita c = new Cita(LocalDateTime.of(2025, 11, 5, 10, 0), p, dr, "Chequeo anual");
20
21         System.out.println(c);
22     }
23 }
24 }
```
- Output Window:** Shows the build log for "Resolucion-TP5 (run)".

```
run:
==> Ejercicio 9 <-- Cita / Paciente / Profesional ==
Cita{fechaHora=2025-11-05T10:00, paciente=Paciente{nombre='Andrea Ruiz', dni='40.222.333'}, profesiona
BUILD SUCCESSFUL (total time: 0 seconds)
```
- Status Bar:** Shows "6354/648.0MB" and "25:1 INS".

## 10. CuentaBancaria - ClaveSeguridad - Titular

- a. Composición: CuentaBancaria → ClaveSeguridad
- b. Asociación bidireccional: CuentaBancaria ↔ Titular

Clases y atributos:

- i. CuentaBancaria: cbu, saldo
- ii. ClaveSeguridad: codigo, ultimaModificacion
- iii. Titular: nombre, dni.

Objetivo Modelar una cuenta bancaria que contiene su clave (dato interno, parte) y está asociada a un titular. La cuenta crea su clave (composición) y conoce a su titular (asociación).

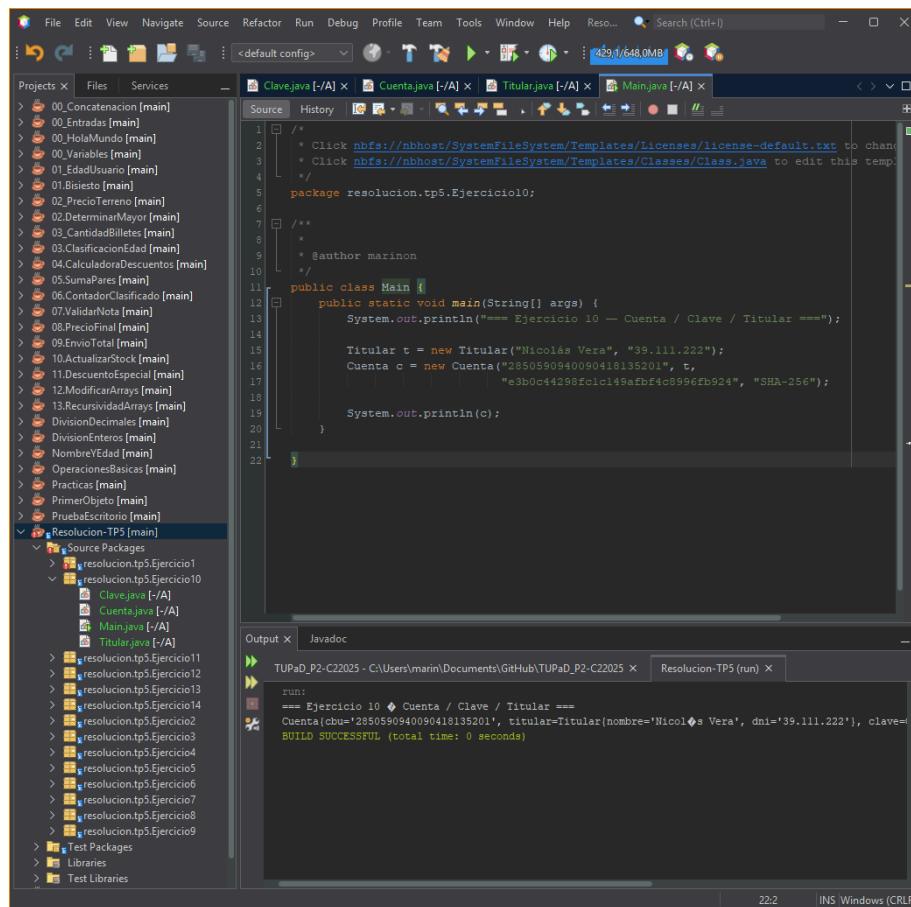
Diagrama UML

Titular 1 —————> 1 Cuenta ◆———— 1 Clave  
 (asociación uni) (composición)

Relaciones y dirección

- Cuenta–Clave: Composición (1:1), unidireccional desde Cuenta.
- Cuenta–Titular: Asociación (1:1), unidireccional desde Cuenta.

Implementación:



```

 1  /*
 2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 4  */
 5  package resolucion.tp5.Ejercicio10;
 6
 7  /**
 8  *
 9  * @author marinon
10  */
11 public class Main {
12     public static void main(String[] args) {
13         System.out.println("== Ejercicio 10 == Cuenta / Clave / Titular ==");
14
15         Titular t = new Titular("Nicolás Vera", "39.111.222");
16         Cuenta c = new Cuenta("2850590940090418135201", t,
17                               "e3b0c4298fc1c149afbf4c8996fb924", "SHA-256");
18
19         System.out.println(c);
20     }
21 }
  
```

## Ejercicios de Dependencia de Uso

## 11. Reproductor - Canción - Artista

- a. Asociación unidireccional: Canción → Artista
  - b. Dependencia de uso: Reproductor.reproducir(Cancion)

## Clases y atributos:

- i. Canción: titulo.
  - ii. Artista: nombre, genero.
  - iii. Reproductor->método: void reproducir(Cancion cancion)

Objetivo: Modelar un reproductor que reproduce canciones sin almacenarlas como estado. Una canción conoce a su artista.

## Diagrama UML

Cancion —————> Artista (asociación unidireccional 1:1)

Reproductor —————> reproducir(Cancion) (dependencia de uso)

## Tipo de relación y dirección

- Canción–Artista: Asociación 1:1. Unidireccional
  - Reproductor–Canción: Dependencia de uso (parámetro de método).

## Implementación:

The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help, Reso... (Search (Ctrl+I))
- Projects View:** Shows multiple Java projects: 00\_Concatenacion [main], 00\_Entradas [main], 00\_HolaMundo [main], 00\_Variables [main], 01\_Eddusuario [main], 01\_Bisiesto [main], 02\_PrecioTercero [main], 02\_DeterminarMayor [main], 03\_CantidadBilletes [main], 03\_ClasicacionEdad [main], 04\_CalculadoraDescuentos [main], 05\_SumaPares [main], 06\_ContadorClasificado [main], 07\_ValidarNota [main], 08\_PrecioFinal [main], 09\_EnvioTotal [main], 10\_ActualizarStock [main], 11\_DescuentoEspecial [main], 12\_ModificarArrays [main], 13\_RecursividadArrays [main], 14\_DivisionDecimales [main], 15\_DivisionEnteros [main], 16\_NombreyEdad [main], 17\_OperacionesBasicas [main], 18\_Practicas [main], 19\_PrimerObjeto [main], 20\_ProblemaEscritorio [main], and Resolucion-TP5 [main].
- Source Editor:** The current file is Main.java, which contains the following code:

```
/*
 * Click nbfs://nbshost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbshost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package resolucion.tp5.Ejercicio11;

/**
 *
 * @author marinon
 */
public class Main {
    public static void main(String[] args) {
        System.out.println("== Ejercicio 11 == Reproductor / Cancion / Artista ==");

        Artista a = new Artista("Gustavo Cerati", "Rock");
        Cancion c = new Cancion("Crimen", a);

        Reproductor r = new Reproductor();
        r.reproducir(c); // dependencia de uso: parámetro del método
    }
}
```
- Output View:** Shows the terminal output of the run command:

```
TUPaD_P2-C22025 - C:\Users\marin\Documents\GitHub\TUPaD_P2-C22025 x Resolucion-TP5 (run) x

run:
== Ejercicio 11 == Reproductor / Cancion / Artista ==
? Reproduciendo: Crimen
    Artista: Artista:nombre='Gustavo Cerati', genero='Rock'
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 12. Impuesto - Contribuyente - Calculadora

- a. Asociación unidireccional: Impuesto → Contribuyente
- b. Dependencia de uso: Calculadora.calcular(Impuesto)

Clases y atributos:

- i. Impuesto: monto.
- ii. Contribuyente: nombre, cuil.
- iii. Calculadora->método: void calcular(Impuesto impuesto)

Objetivo: Modelar una calculadora de impuestos que calcula sobre un impuesto recibido como parámetro (sin guardararlo). Un impuesto conoce al contribuyente al que pertenece.

Diagrama UML

Impuesto —————> Contribuyente (asociación unidireccional 1:1)

Calculadora —————> calcular(Impuesto) (dependencia de uso)

Tipo de relación y dirección

- Impuesto–Contribuyente: Asociación 1:1. Unidireccional.
- Calculadora–Impuesto: Dependencia de uso (parámetro de método).

Implementación

```

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Reso... Search (Ctrl+I)
Projects Files Services
Calculator.java [-A] x Contribuyente.java [-A] x Impuesto.java [-A] x Main.java [-A] x
Source History Tools Find Replace Go To Type In Editor Run Build Run Configuration...
1 /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license.
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template.
4  */
5 package resolucion.tp5.Ejercicio12;
6
7 /**
8  *
9  * @author marinon
10 */
11 public class Main {
12     public static void main(String[] args) {
13         System.out.println("== Ejercicio 12 == Impuesto / Contribuyente / Calculadora");
14
15         Contribuyente cont = new Contribuyente("Laura Pérez", "27-40123456-9");
16         Impuesto imp = new Impuesto(100000.0, cont);
17
18         Calculadora calc = new Calculadora();
19         calc.calcular(imp); // dependencia de uso
20     }
21 }

```

Output X Javadoc

```

TUPaD_P2-C22025 - C:\Users\marin\Documents\GitHub\TUPaD_P2-C22025 x Resolucion-TP5 (run) x
run:
== Ejercicio 12 == Impuesto / Contribuyente / Calculadora ===
Calculando para Contribuyente[nombre='Laura Pérez', cuil='27-40123456-9']
Base: 100000,00 IVA(21%): 21000,00 Total: 121000,00
BUILD SUCCESSFUL (total time: 0 seconds)

```

Finished building Resolucion-TP5 (run).

## 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:

- i. CódigoQR: valor.
- ii. Usuario: nombre, email.
- iii. GeneradorQR->método: void generar(String valor, Usuario usuario)

Objetivo: Modelar un generador de códigos QR que crea un código nuevo dentro del método generar() a partir de un valor y un usuario.

El QR conoce al usuario, pero el generador no conserva ninguna referencia (dependencia de creación).

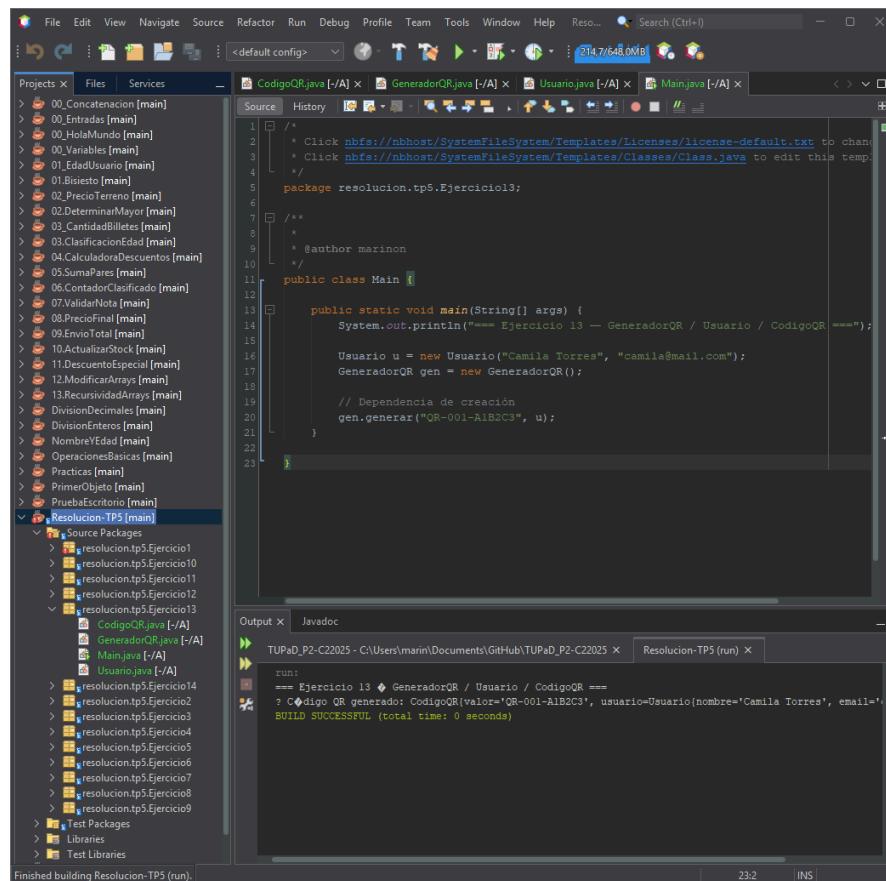
## Diagrama UML

CódigoQR —————> Usuario (asociación unidireccional 1:1)  
 GeneradorQR —————> generar(String, Usuario) (dependencia de creación)

## Tipo de relación y dirección

- CódigoQR–Usuario: Asociación unidireccional 1:1.
- GeneradorQR–CódigoQR: Dependencia de creación (crea, pero no guarda).

## Implementación



```

File Edit View Navigate Source Refactor Run Debug Profile Team Window Help Reso... Search (Ctrl+)
Projects X Files Services CodigoQR.java [-/A] X GeneradorQR.java [-/A] X Usuario.java [-/A] X Main.java [-/A] X
Source History ...
1 /**
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license.
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template.
4 */
5 package resolucion.tp5.Ejercicio13;
6
7 /**
8  * @author marinon
9 */
10 public class Main {
11
12     public static void main(String[] args) {
13         System.out.println("==== Ejercicio 13 — GeneradorQR / Usuario / CódigoQR ====");
14
15         Usuario u = new Usuario("Camila Torres", "camila@mail.com");
16         GeneradorQR gen = new GeneradorQR();
17
18         // Dependencia de creación
19         gen.generar("QR-001-AlB2C3", u);
20     }
21 }
  
```

The screenshot shows an IDE interface with several tabs open: CodigoQR.java, GeneradorQR.java, Usuario.java, and Main.java. The Main.java tab is active, displaying the provided Java code. The code defines a Main class with a main method. Inside the main method, it prints a welcome message, creates a Usuario object named 'u' with the name 'Camila Torres' and email 'camila@mail.com', and creates a GeneradorQR object named 'gen'. Finally, it calls the 'generar' method on 'gen' with the string 'QR-001-AlB2C3' and the 'u' object. In the bottom right corner of the IDE, there is a terminal window showing the output of the run command, which includes the printed message and the generated QR code string.

#### 14. EditorVideo - Proyecto - Render

- Asociación unidireccional: Render → Proyecto
- Dependencia de creación: EditorVideo.exportar(String, Proyecto)

Clases y atributos:

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

Objetivo: Simular un editor de video que exporta proyectos en distintos formatos.

El editor crea un objeto Render dentro del método exportar(), el cual está asociado a un Proyecto.

La dependencia es de creación, no se conserva el objeto generado.

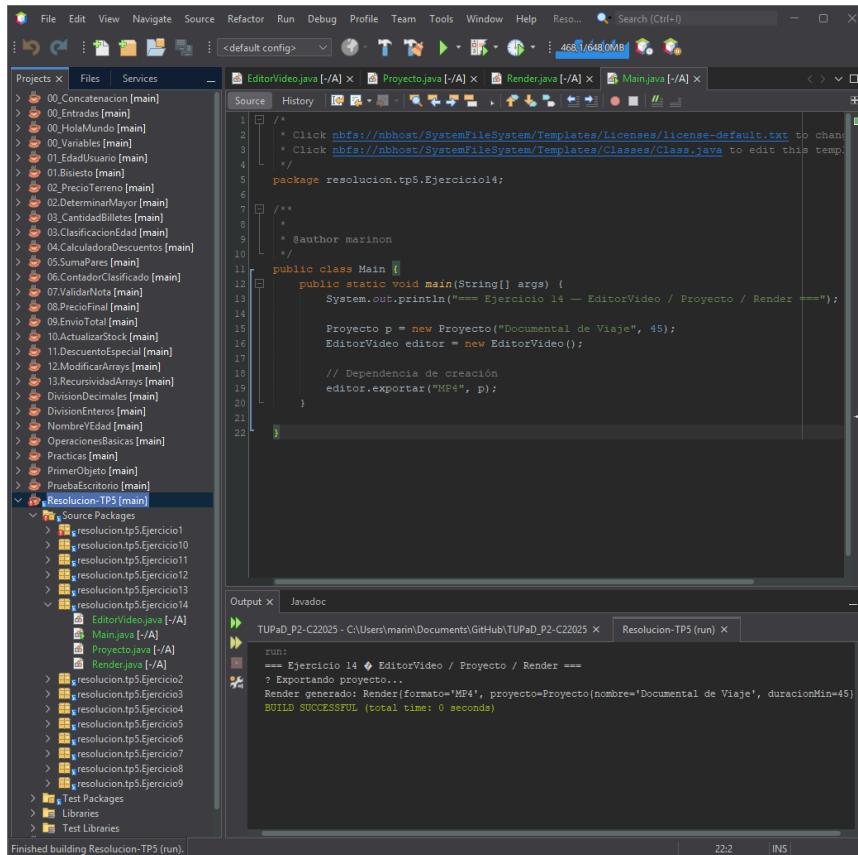
#### Diagrama UML

Render —————> Proyecto (asociación unidireccional 1:1)  
 EditorVideo —————> exportar(String, Proyecto) (dependencia de creación)

Tipo de relación y dirección.

- Render–Proyecto: Asociación unidireccional 1:1.
- EditorVideo–Render: Dependencia de creación.

#### Implementación



```

 1  /*
 2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license.
 3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template.
 4  */
 5  package resolucion.tp5.Ejercicio14;
 6
 7  /**
 8  *
 9  * @author marinon
10  */
11 public class Main {
12     public static void main(String[] args) {
13         System.out.println("==== Ejercicio 14 — EditorVideo / Proyecto / Render ====");
14
15         Proyecto p = new Proyecto("Documental de Viaje", 45);
16         EditorVideo editor = new EditorVideo();
17
18         // Dependencia de creación
19         editor.exportar("MP4", p);
20     }
21 }
  
```

## Conclusión

A lo largo del presente trabajo práctico se desarrollaron 14 ejercicios que permitieron explorar y aplicar las distintas formas de vinculación entre clases en el paradigma de la Programación Orientada a Objetos, representadas mediante diagramas UML.

En las primeras consignas (Ejercicios 1 al 4), se abordaron las relaciones de asociación, agregación y composición, a partir de ejemplos concretos como el vínculo entre un pasaporte y su foto, un celular y su batería, o un cliente y su banco. Estas actividades ayudaron a comprender cómo varía el nivel de dependencia entre los objetos según el tipo de relación.

Por ejemplo, en la composición una clase se responsabiliza totalmente por la existencia de otra (como un pasaporte que crea su propia foto), mientras que en la agregación los objetos pueden existir por separado (como una batería que puede ser retirada o reemplazada de un celular). La asociación, por su parte, implica un conocimiento mutuo entre objetos, sin que haya control sobre el ciclo de vida de uno por parte del otro.

En los ejercicios intermedios (del 5 al 10), se trabajó con relaciones más complejas, incorporando vínculos bidireccionales y combinando distintos tipos de relaciones en un mismo modelo. Casos como el de una computadora con su placa madre y su propietario, o el de una reserva asociada a un cliente y a una mesa, reforzaron la importancia de manejar correctamente las referencias entre clases y respetar el principio de encapsulamiento para mantener la coherencia y claridad del código.

Más adelante, en los Ejercicios 11 y 12, se introdujo el concepto de dependencia de uso, en el cual una clase utiliza otra de forma puntual como parámetro de un método, sin conservarla como atributo. Ejemplos como un reproductor que reproduce una canción o una calculadora que procesa un impuesto mostraron cómo lograr bajo acoplamiento entre objetos, manteniendo independencia funcional.

Finalmente, los últimos ejercicios (13 y 14) abordaron la dependencia de creación, donde una clase instancia otra dentro de un método sin almacenarla. Modelos como el de un generador de códigos QR que crea un objeto asociado a un usuario, o un editor de video que exporta un render vinculado a un proyecto, permitieron reflexionar sobre el rol de ciertos objetos como creadores temporales o "fábricas" dentro del diseño.

En conjunto, este trabajo fue clave para afianzar la capacidad de traducir relaciones UML a código en Java, aplicando los distintos tipos de vínculos según el contexto, prestando atención a la direccionalidad de las relaciones, y organizando los atributos, métodos y constructores de manera clara y coherente.

Además, permitió ejercitarse en varios principios fundamentales de la programación orientada a objetos como el encapsulamiento, la modularidad, la reutilización y el bajo acoplamiento, fortaleciendo así una base sólida para futuros desarrollos en este paradigma.

### Link a Repositorio GitHub:

[https://github.com/MaquiMarinoni/TUPaD\\_P2-C22025.git](https://github.com/MaquiMarinoni/TUPaD_P2-C22025.git)