**PROGRAMACIÓN II**

**Trabajo Práctico 5: Relaciones UML 1 a 1**

# Alumno:

Franco Sarrú

# Link público de GitHub:

<https://github.com/fsarru/Programacion2.git>

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

# Caso Práctico

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

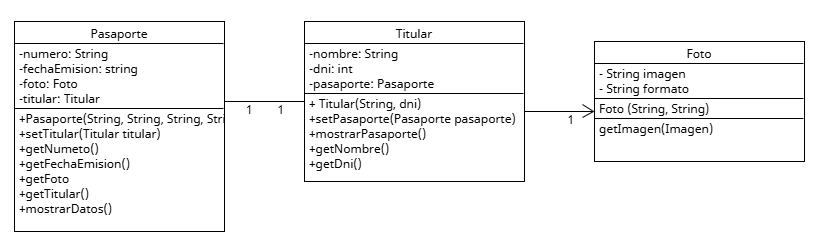
* Diagrama UML
* Tipo de relación (asociación, agregación, composición, dependencia)
* Dirección (unidireccional o bidireccional)
* Implementación de las clases con atributos y relaciones definidas

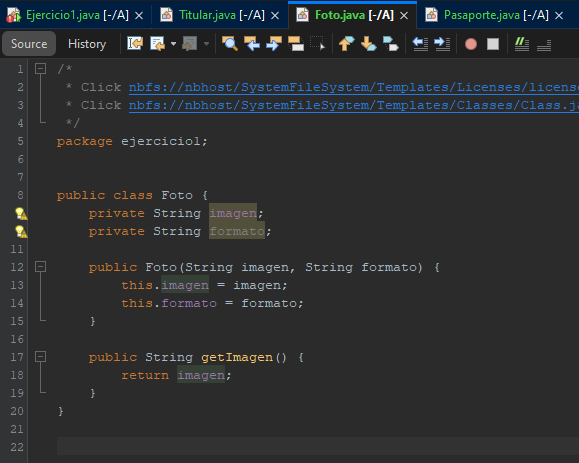
**Ejercicios de Relaciones 1 a 1**

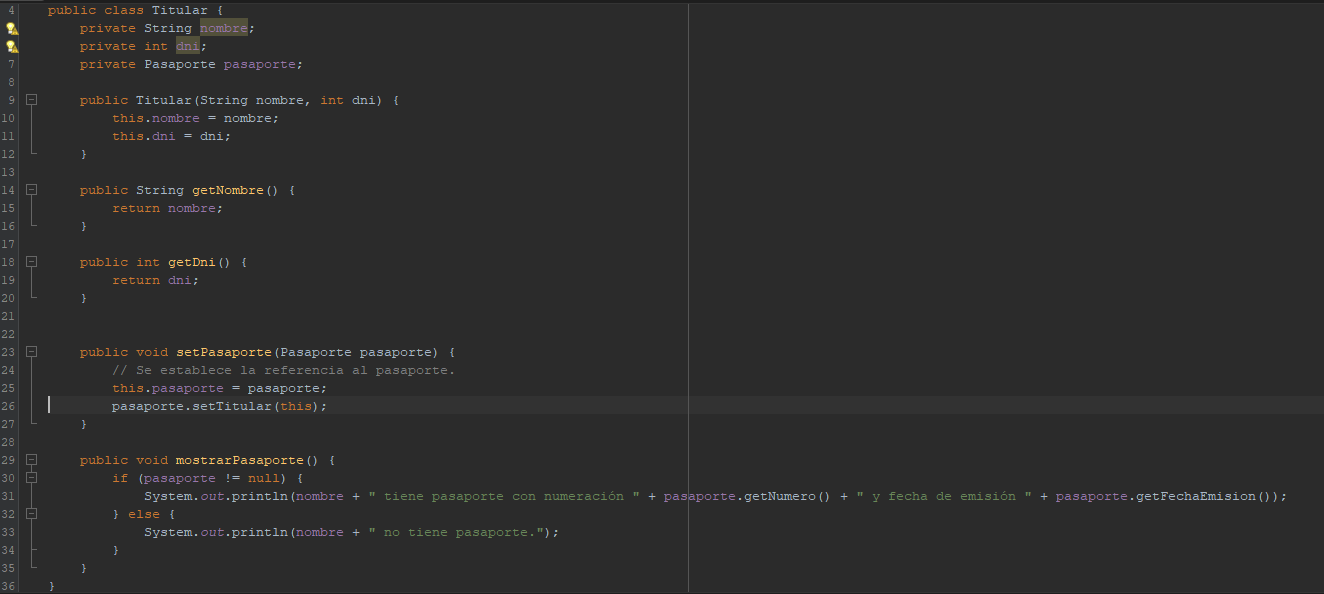
1. Pasaporte - Foto - Titular
   1. Composición: **Pasaporte → Foto**
   2. Asociación bidireccional: **Pasaporte ↔ Titular**

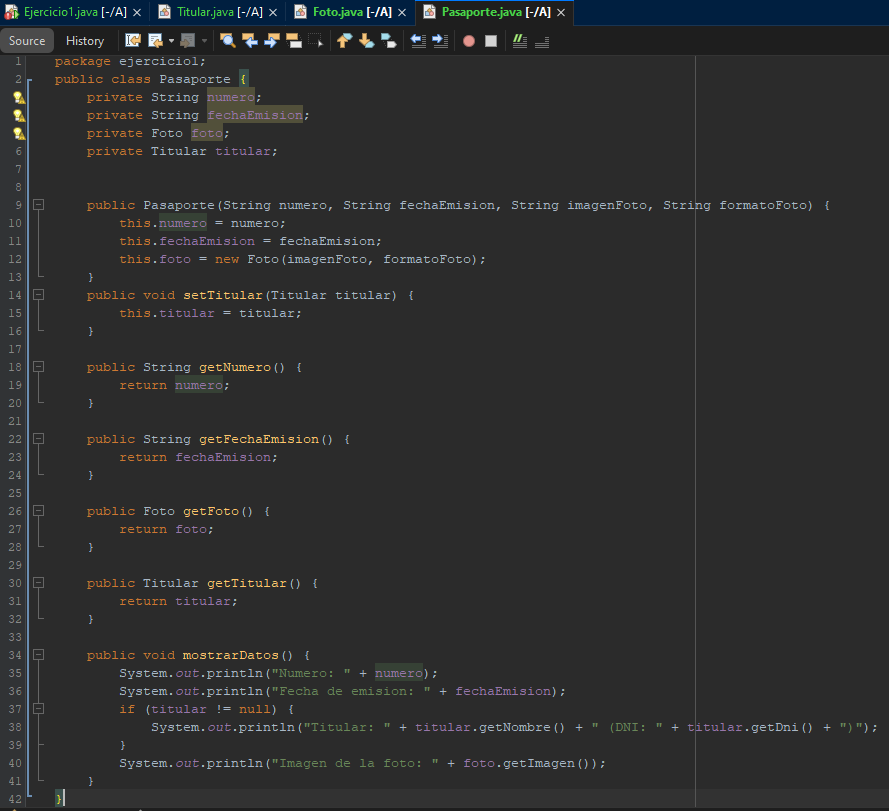
Clases y atributos:

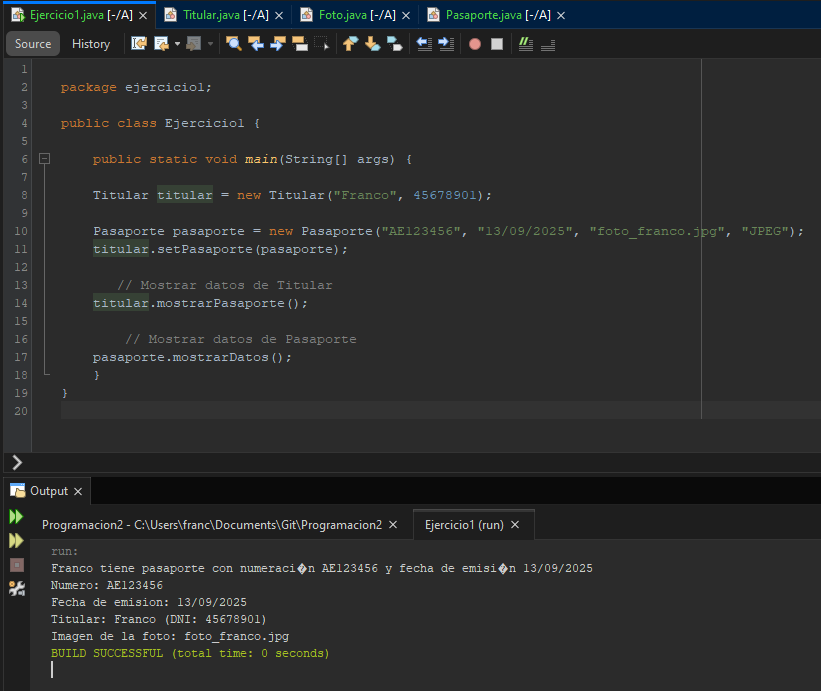
* + 1. Pasaporte: numero, fechaEmision
    2. Foto: imagen, formato
    3. Titular: nombre, dni







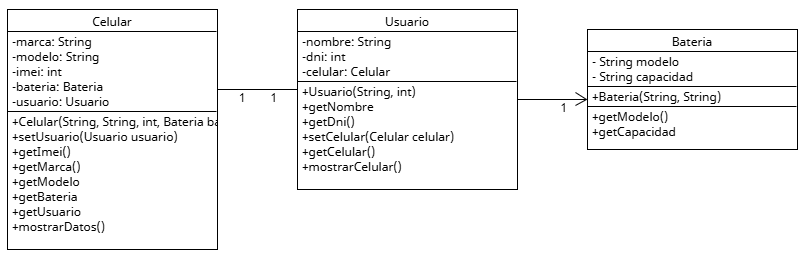


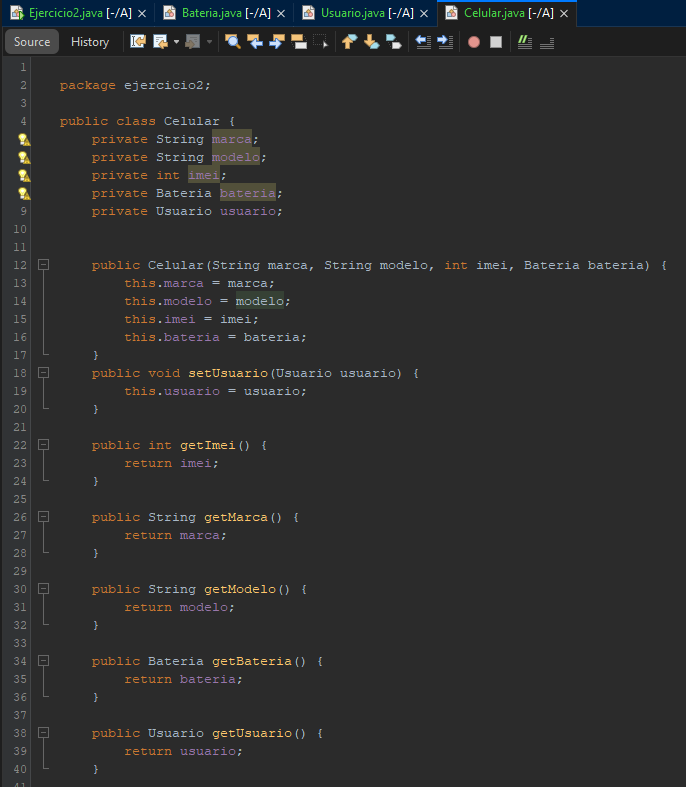


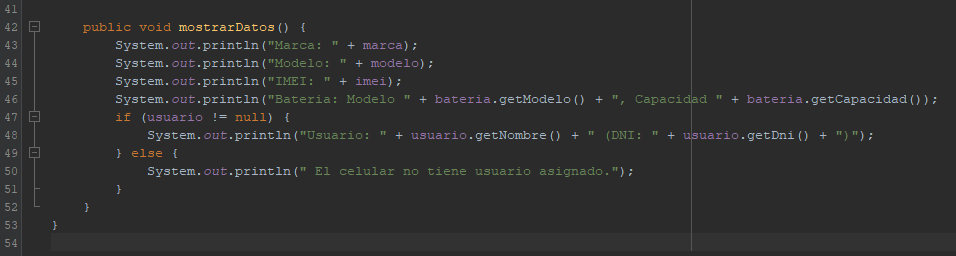
1. Celular - Batería - Usuario
   1. Agregación: **Celular → Batería**
   2. Asociación bidireccional: **Celular ↔ Usuario**

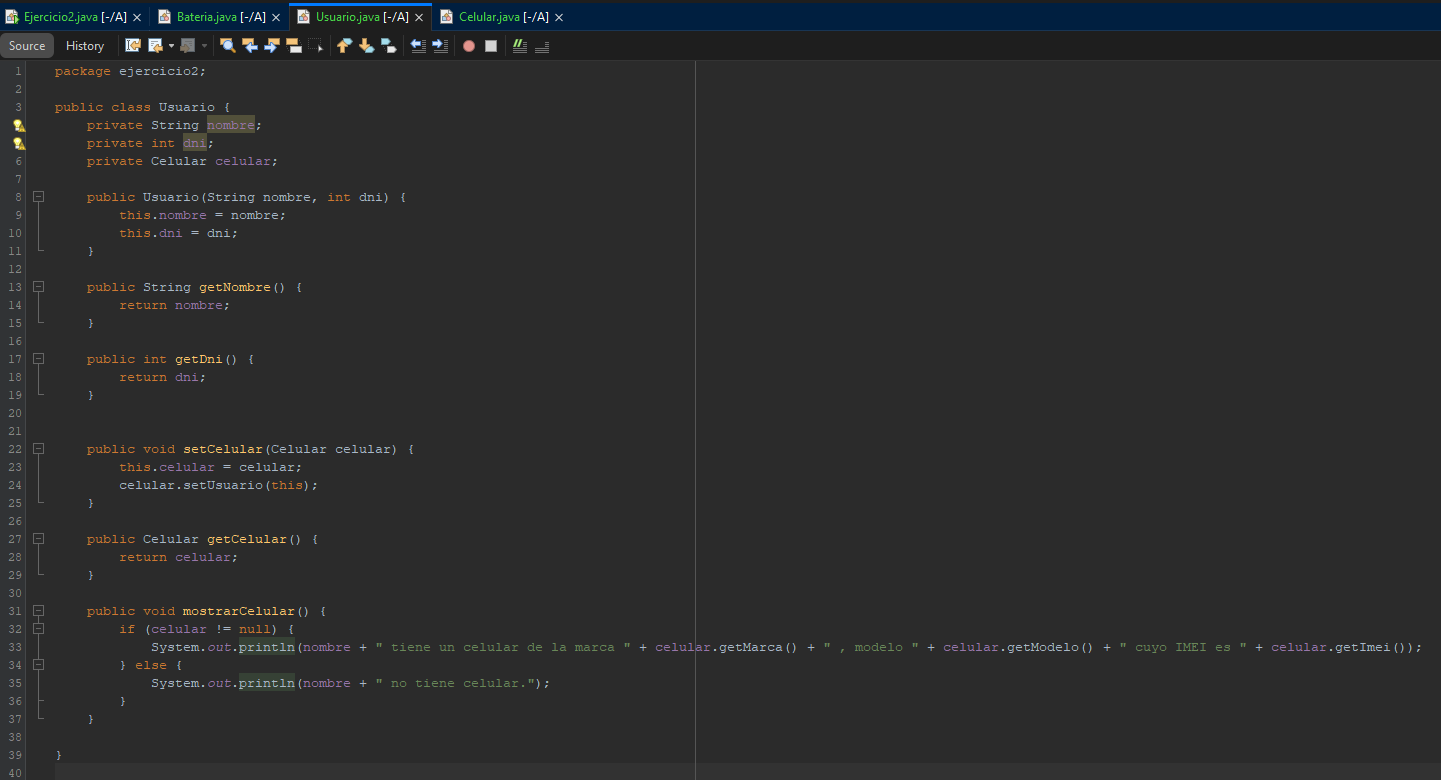
Clases y atributos:

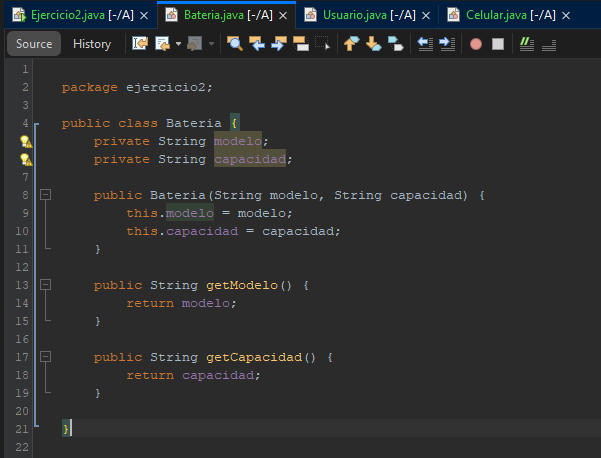
* + 1. Celular: imei, marca, modelo
    2. Batería: modelo, capacidad
    3. Usuario: nombre, dni

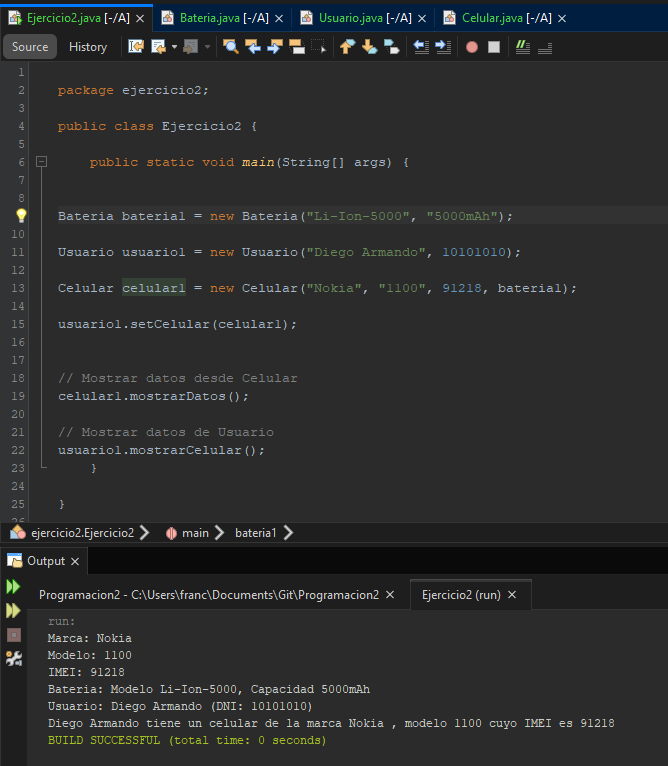










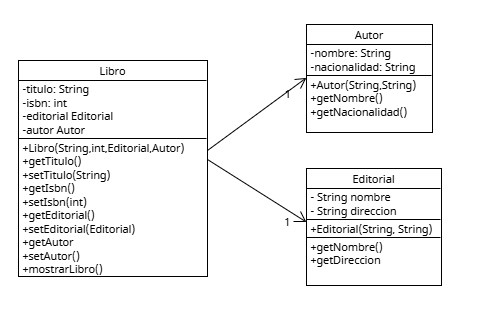


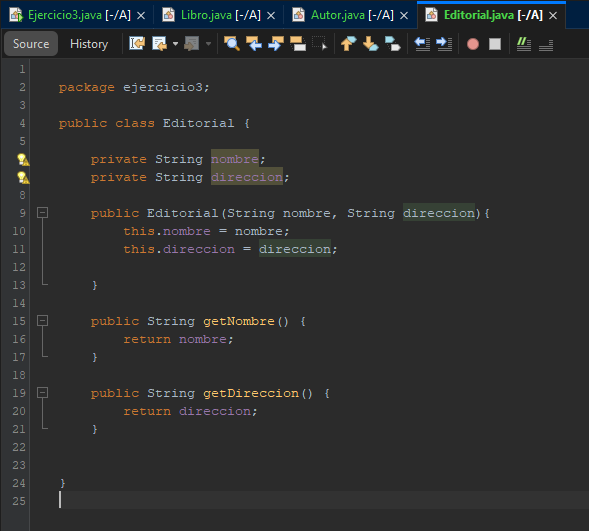
1. Libro - Autor - Editorial
   1. Asociación unidireccional: **Libro → Autor**
   2. Agregación: **Libro → Editorial**

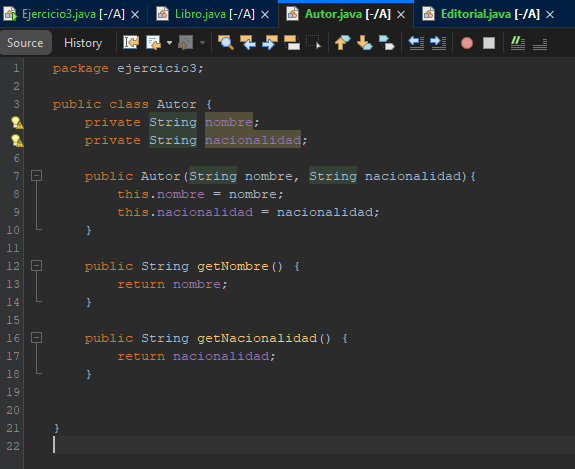
Clases y atributos:

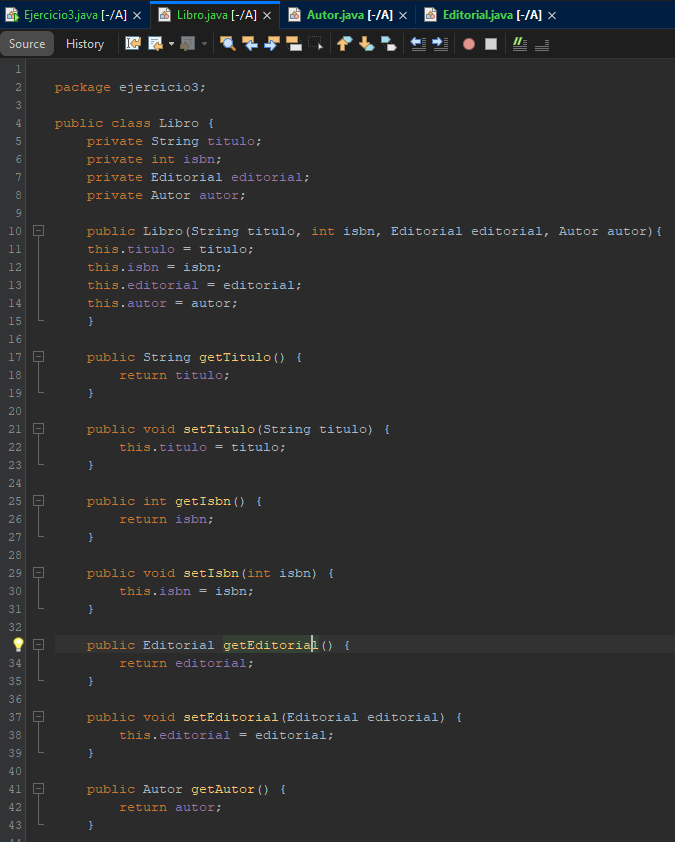
* + 1. Libro: titulo, isbn
    2. Autor: nombre, nacionalidad

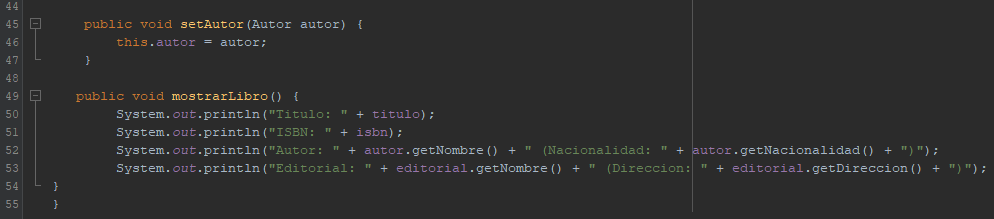
iii. Editorial: nombre, direccion

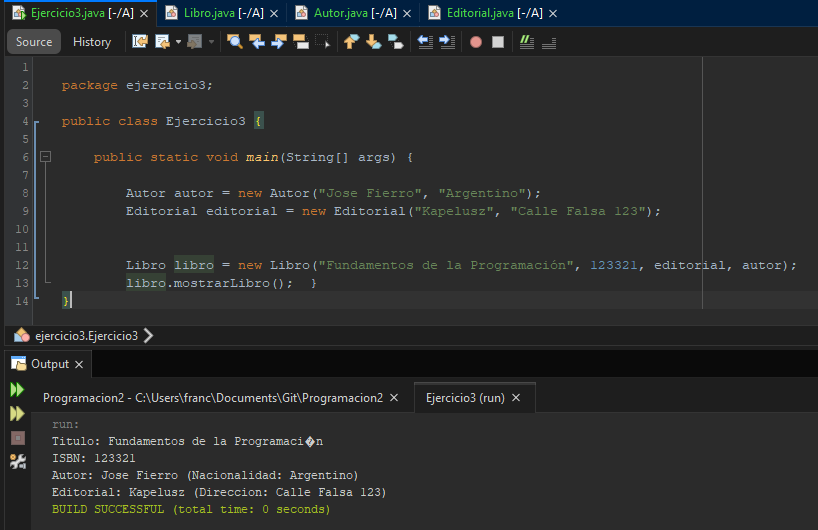








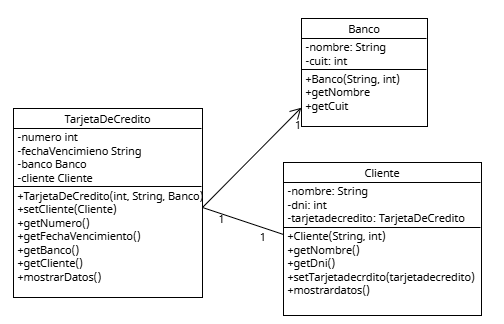


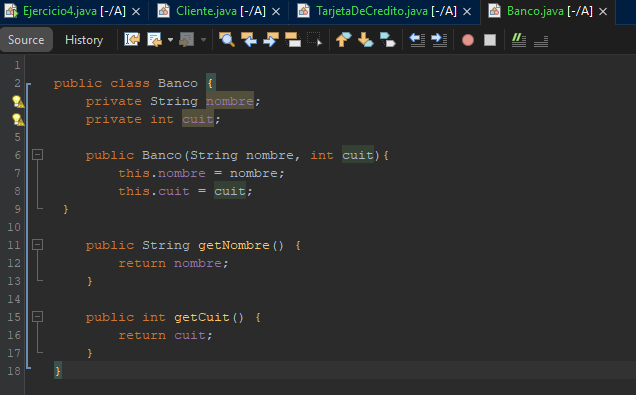


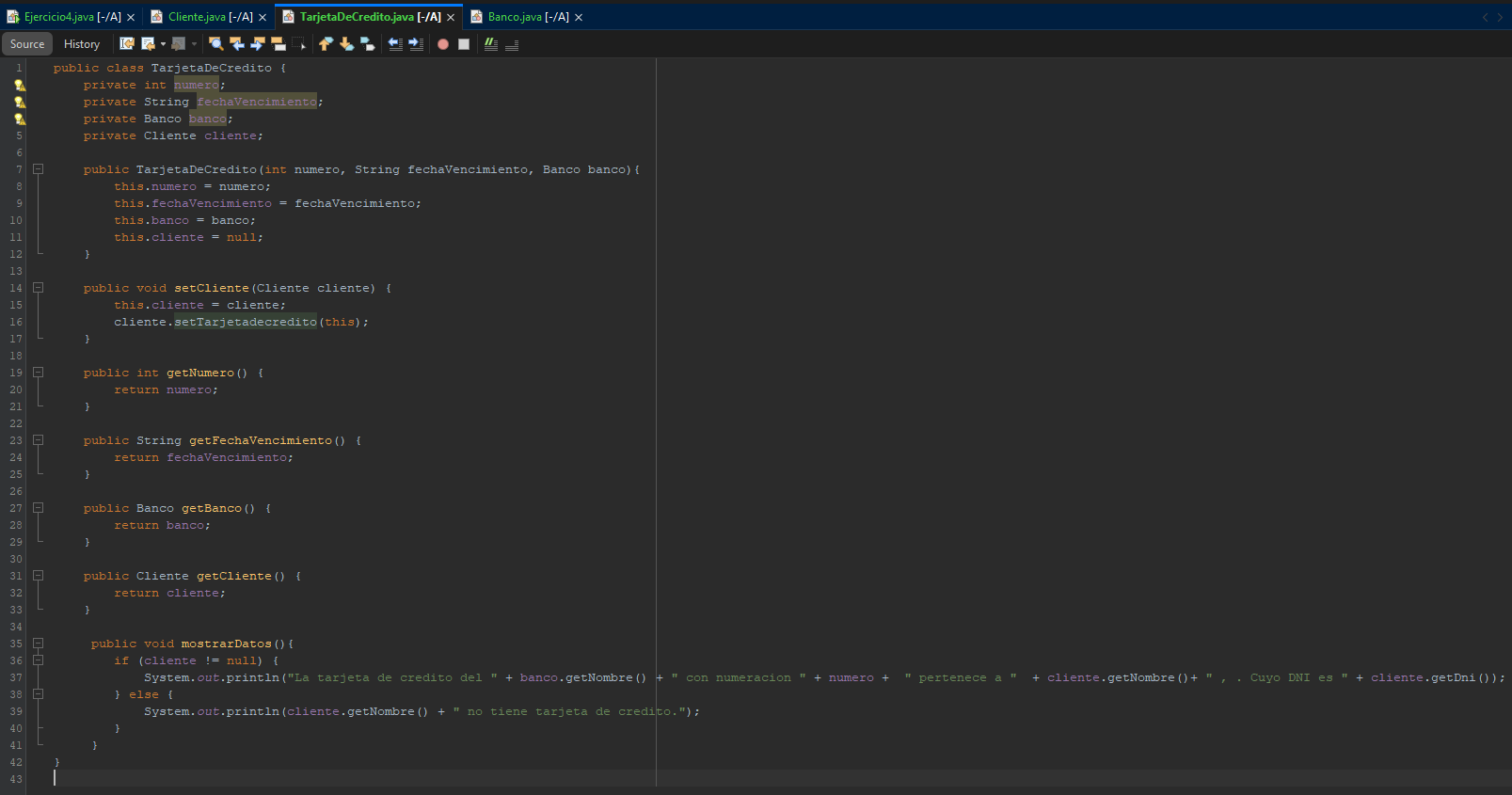
1. TarjetaDeCrédito - Cliente - Banco
   1. Asociación bidireccional: **TarjetaDeCrédito ↔ Cliente**
   2. Agregación: **TarjetaDeCrédito → Banco**

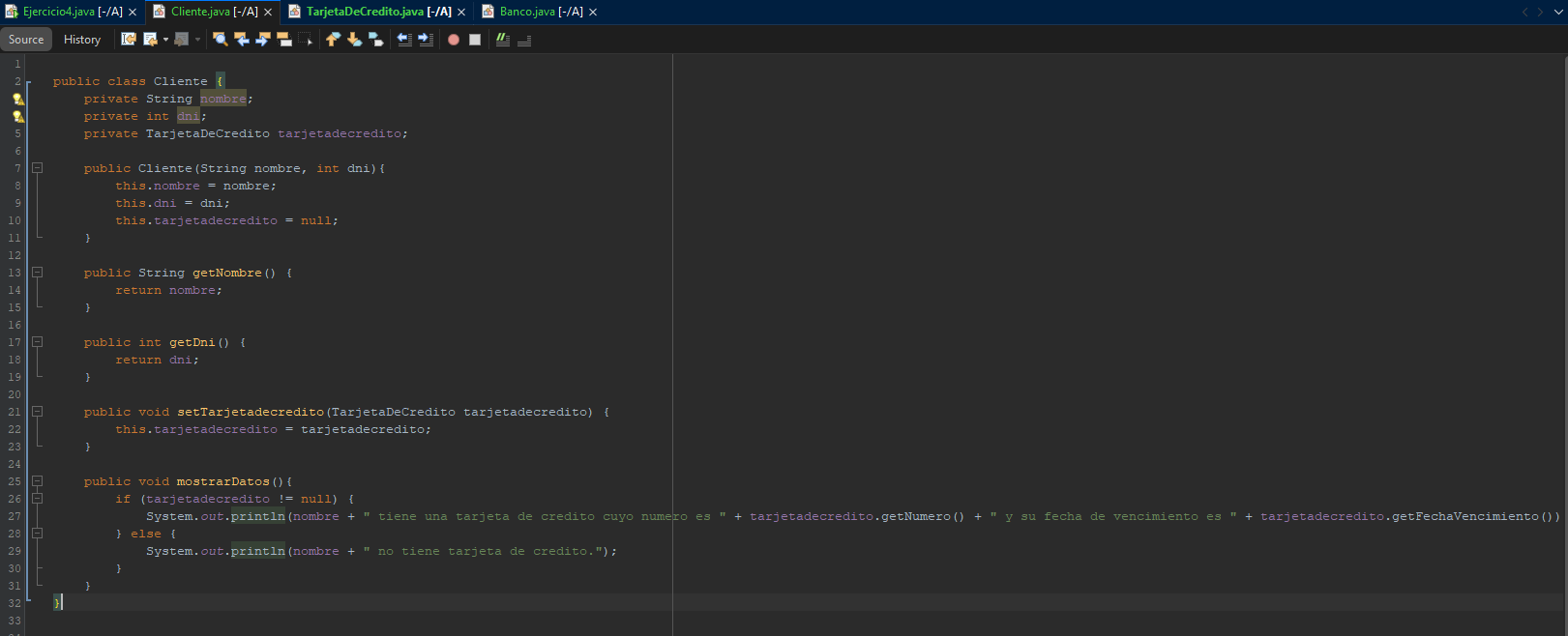
Clases y atributos:

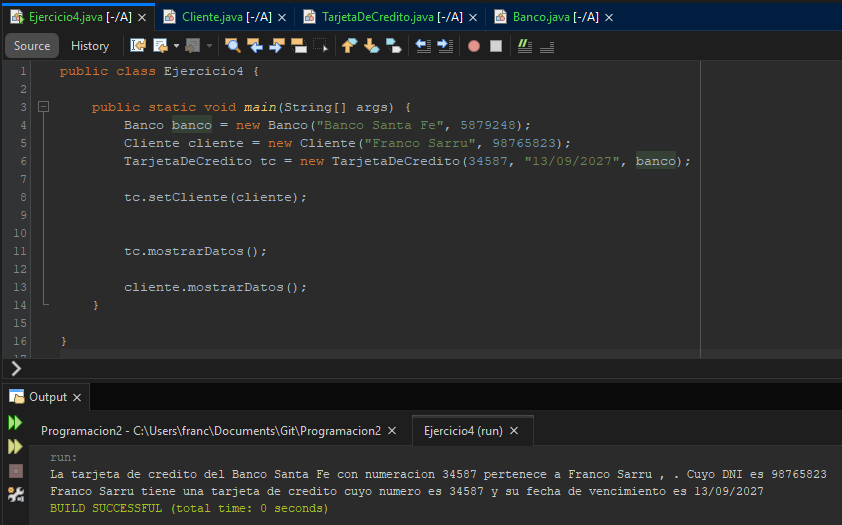
* + 1. TarjetaDeCrédito: numero, fechaVencimiento
    2. Cliente: nombre, dni
    3. Banco: nombre, cuit







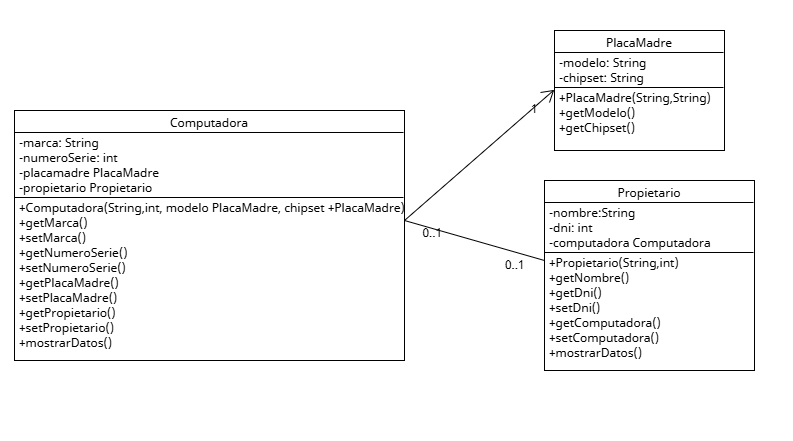


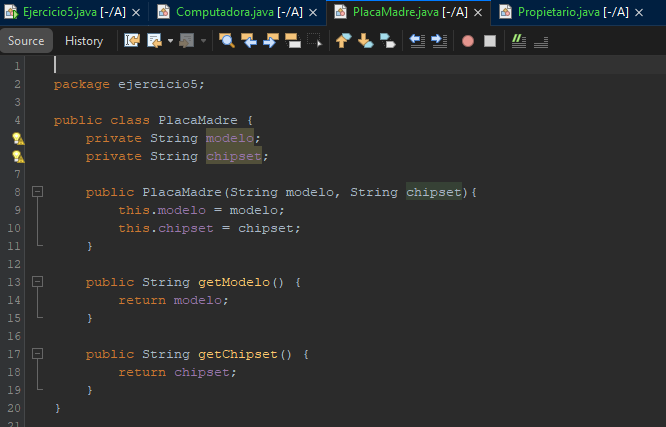


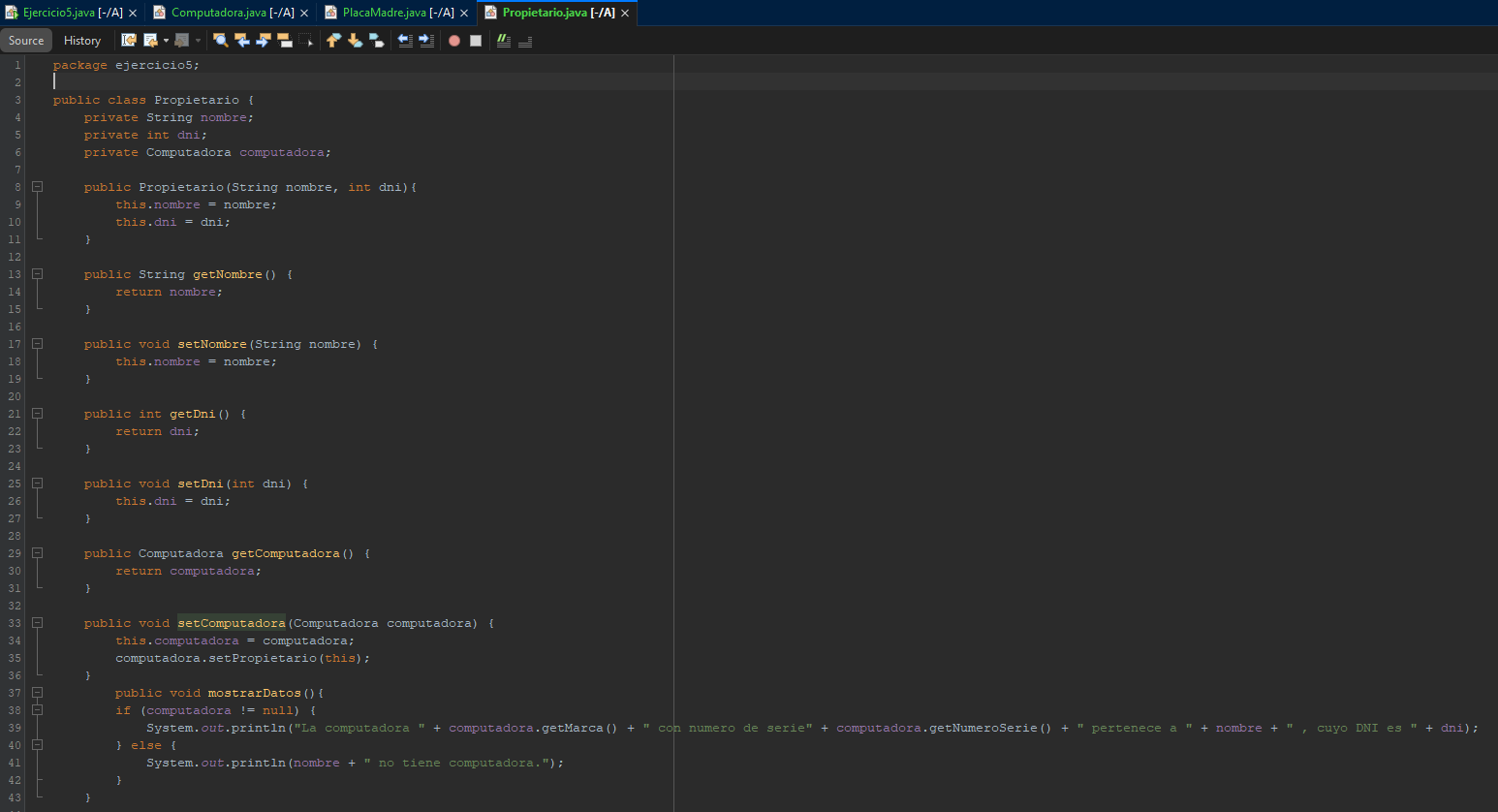
1. Computadora - PlacaMadre - Propietario
   1. Composición: **Computadora → PlacaMadre**
   2. Asociación bidireccional: **Computadora ↔ Propietario**

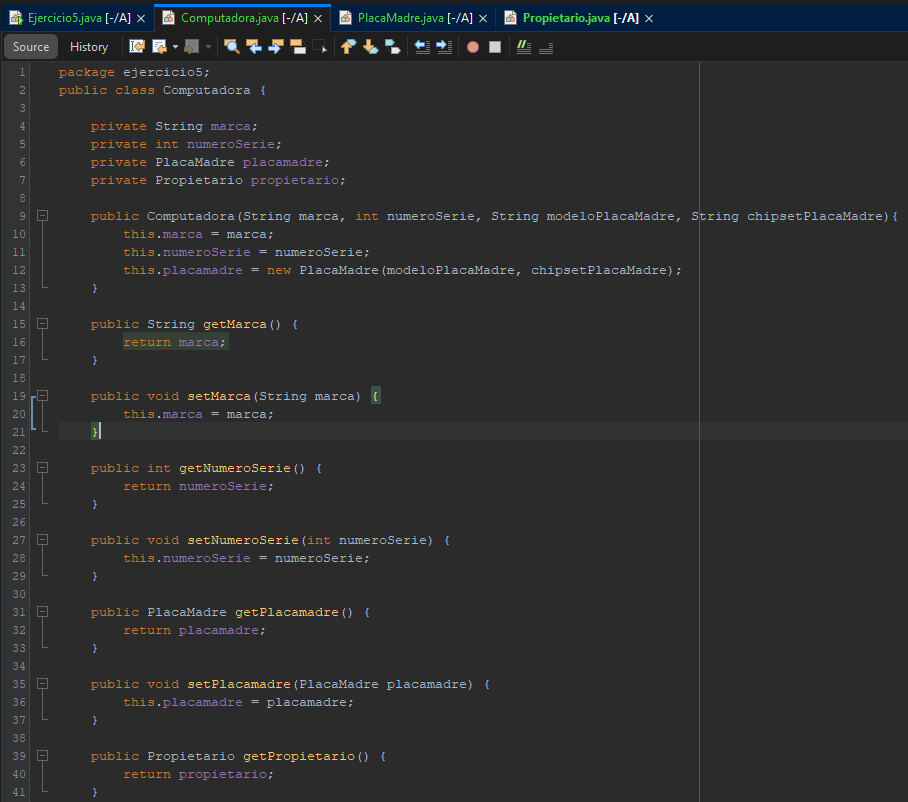
Clases y atributos:

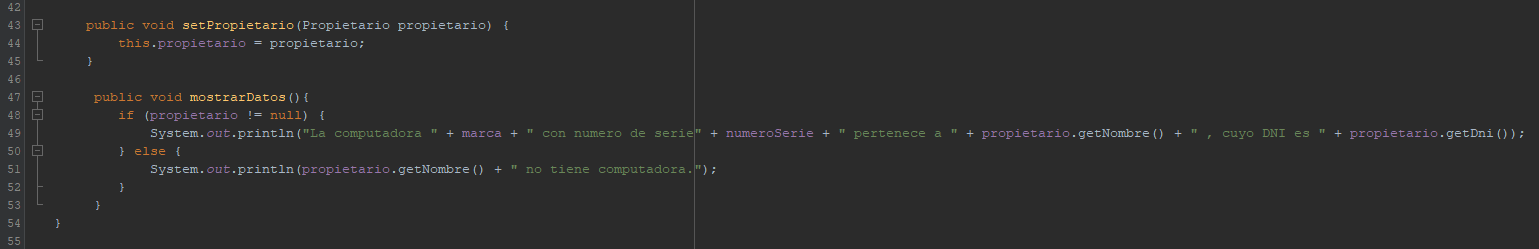
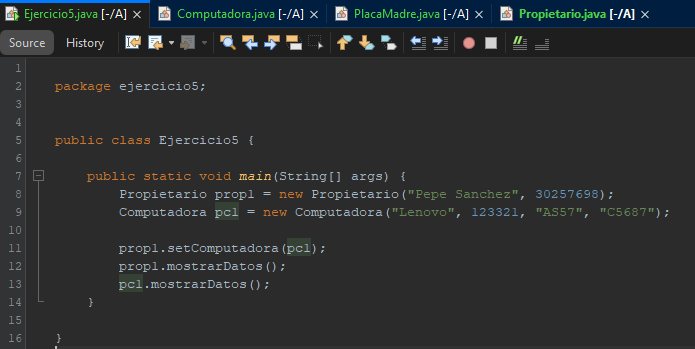
* + 1. Computadora: marca, numeroSerie
    2. PlacaMadre: modelo, chipset
    3. Propietario: nombre, dni







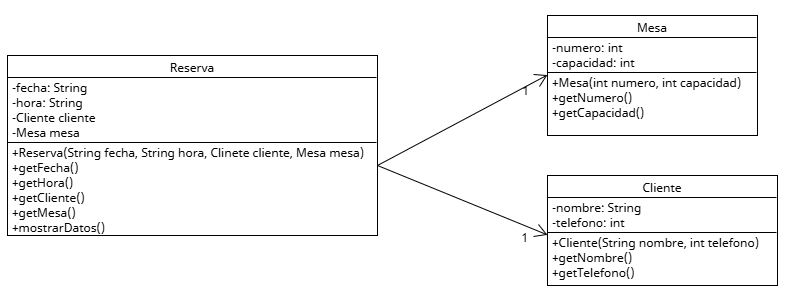


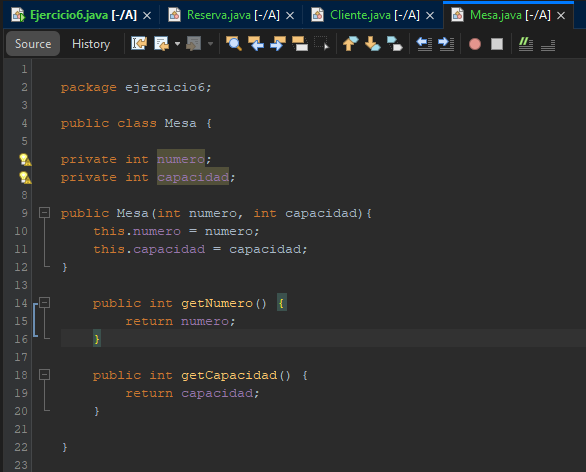
 

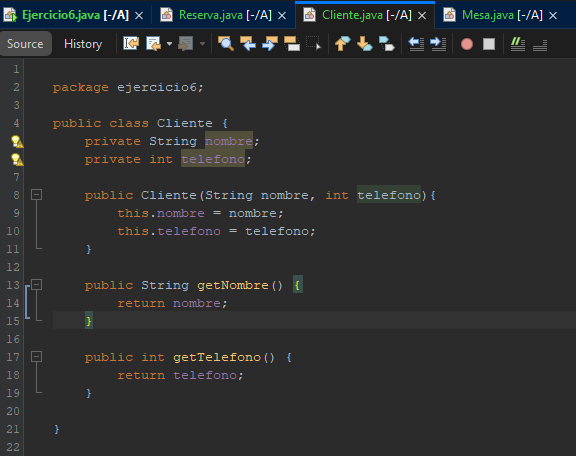
1. Reserva - Cliente - Mesa
   1. Asociación unidireccional: **Reserva → Cliente**
   2. Agregación: **Reserva → Mesa**

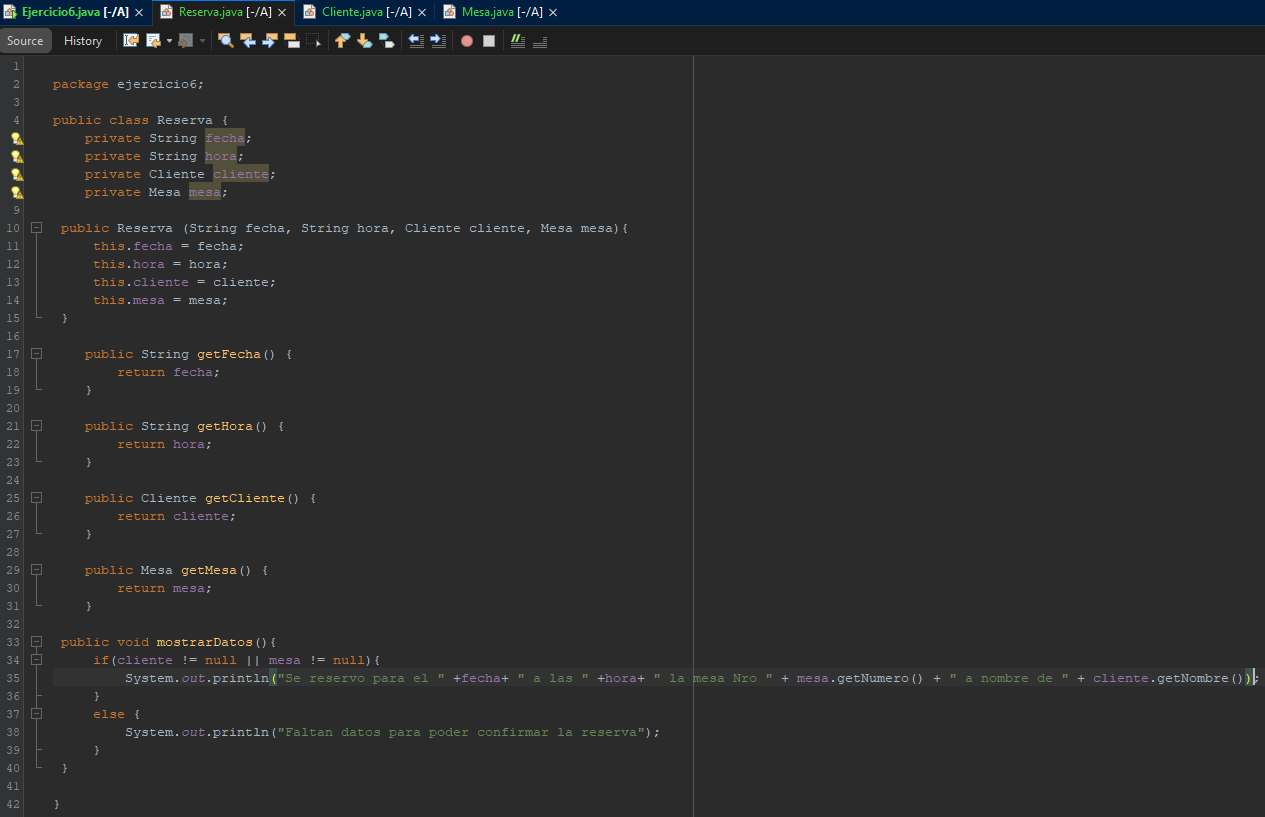
Clases y atributos:

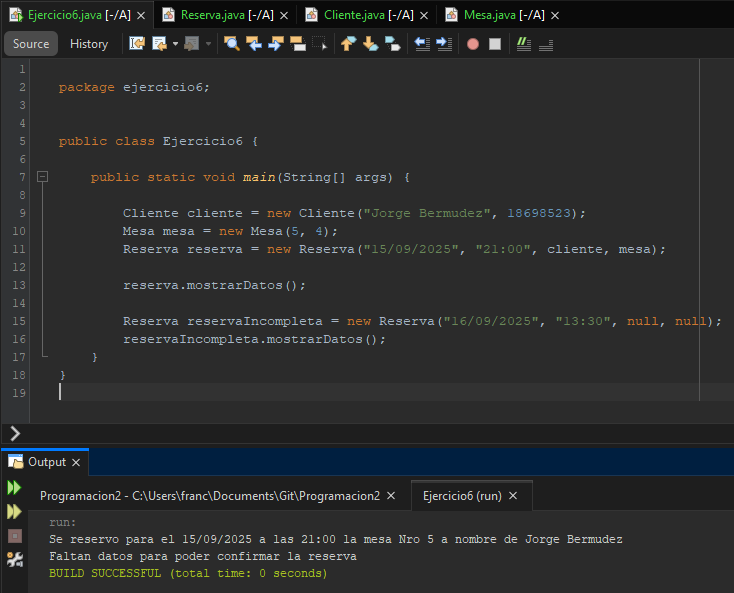
* + 1. Reserva: fecha, hora
    2. Cliente: nombre, telefono
    3. Mesa: numero, capacidad







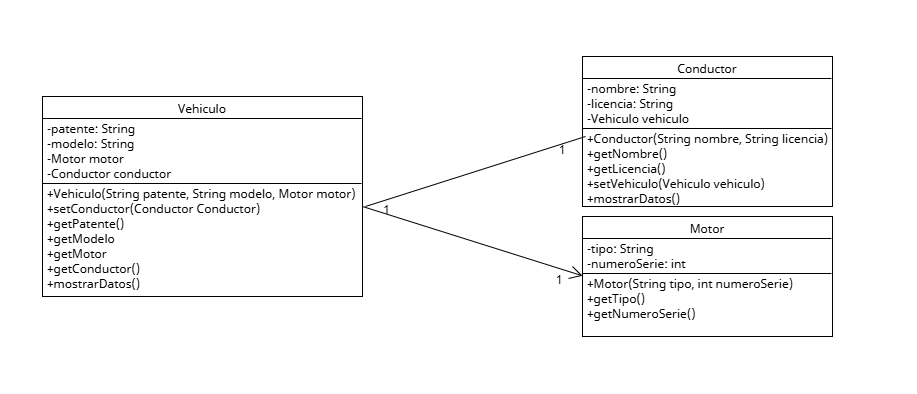


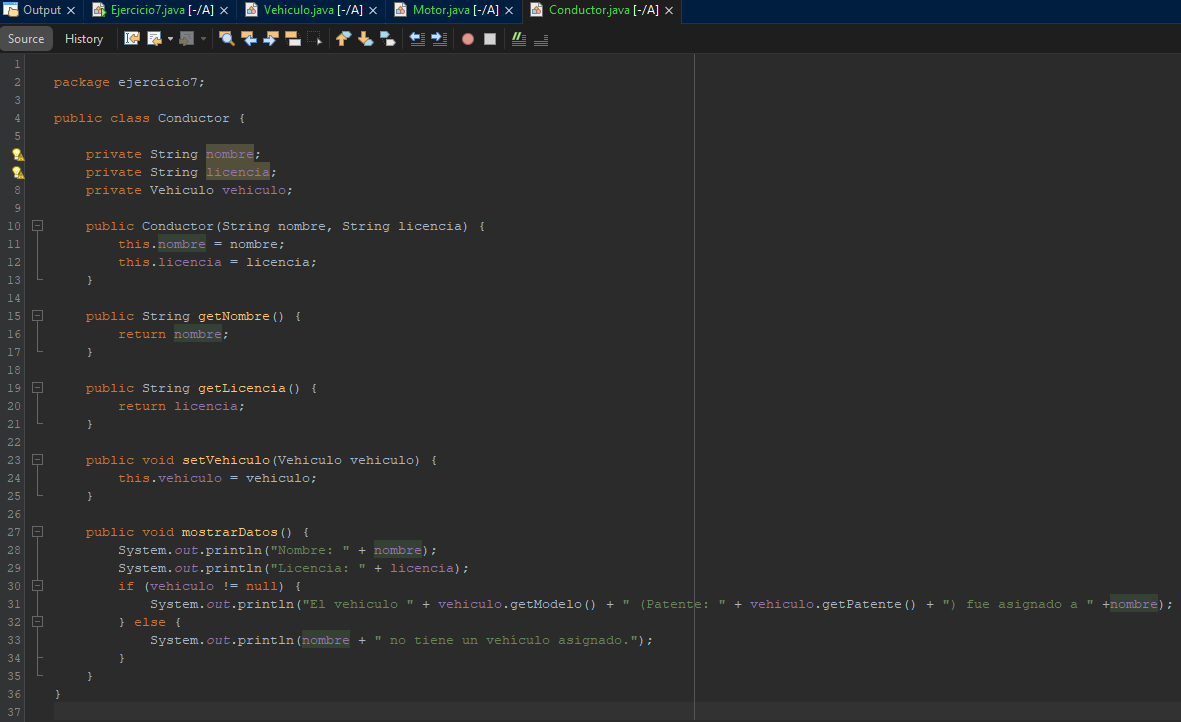


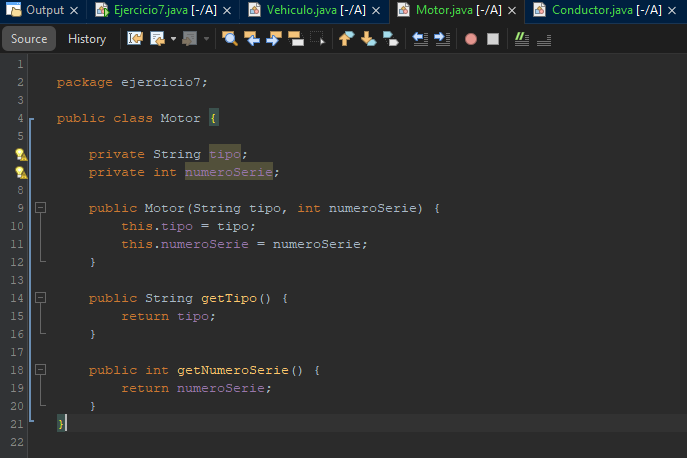
1. Vehículo - Motor - Conductor
   1. Agregación: **Vehículo → Motor**
   2. Asociación bidireccional: **Vehículo ↔ Conductor**

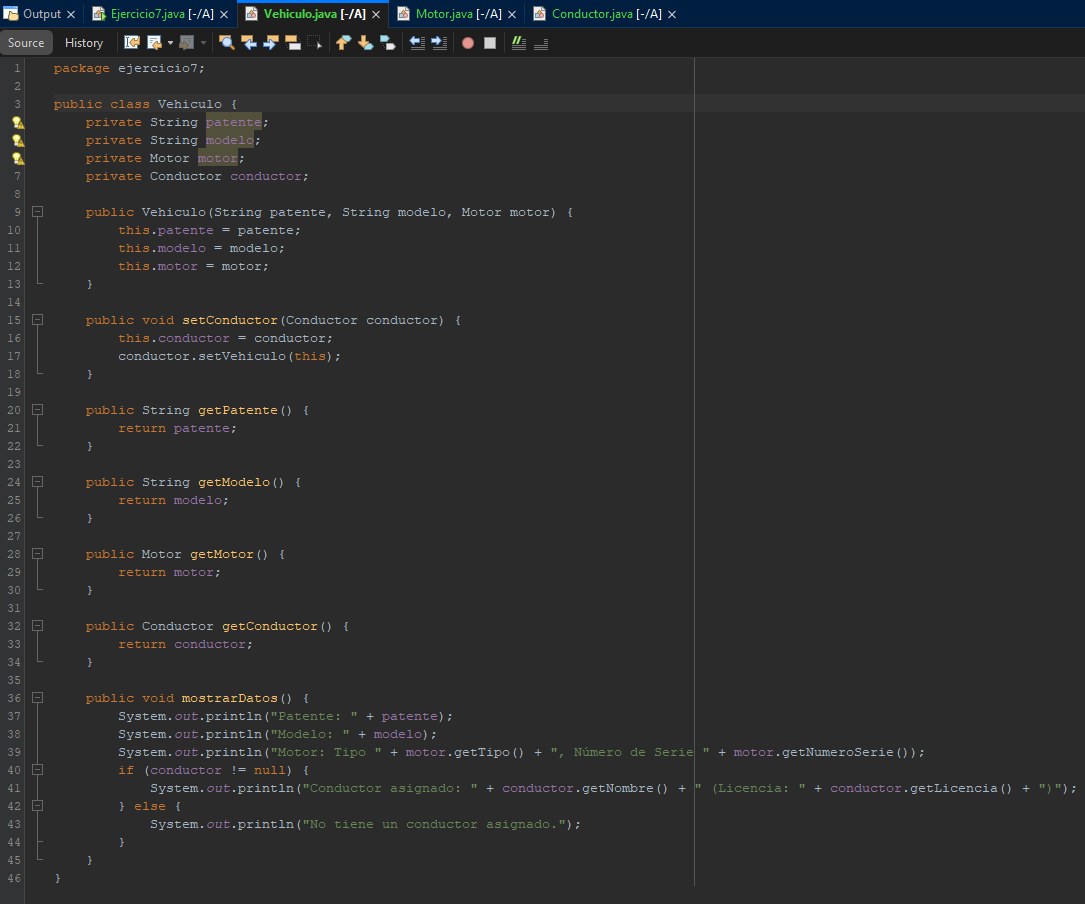
Clases y atributos:

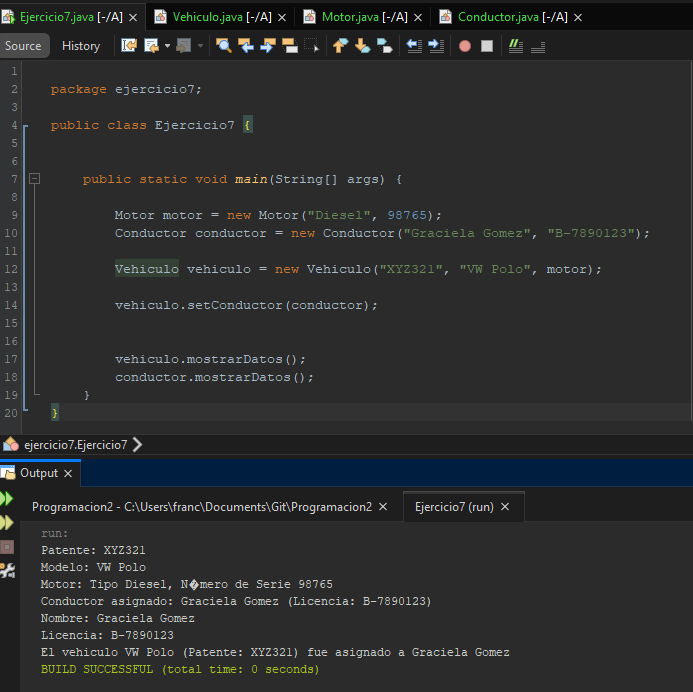
* + 1. Vehículo: patente, modelo
    2. Motor: tipo, numeroSerie
    3. Conductor: nombre, licencia







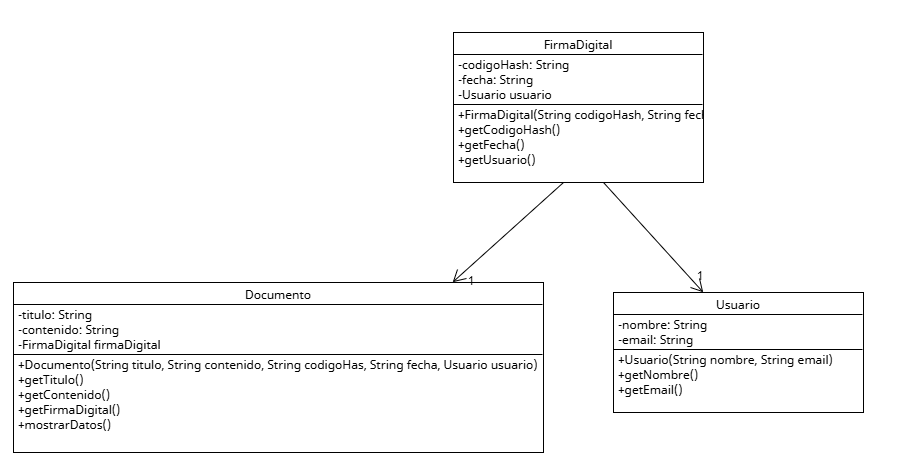


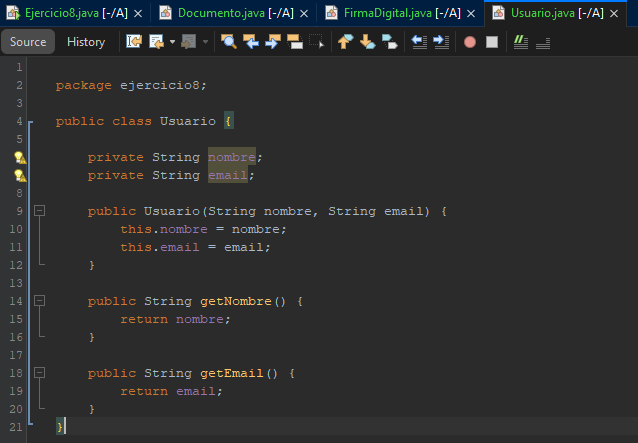


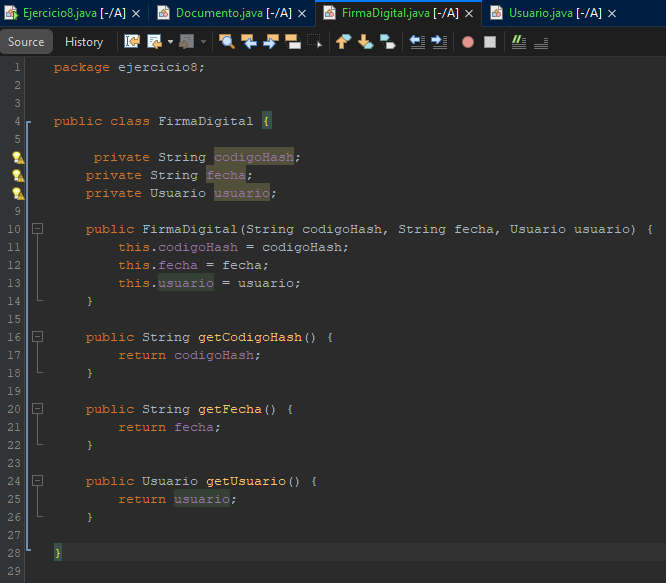
1. Documento - FirmaDigital - Usuario
   1. Composición: **Documento → FirmaDigital**
   2. Agregación: **FirmaDigital → Usuario**

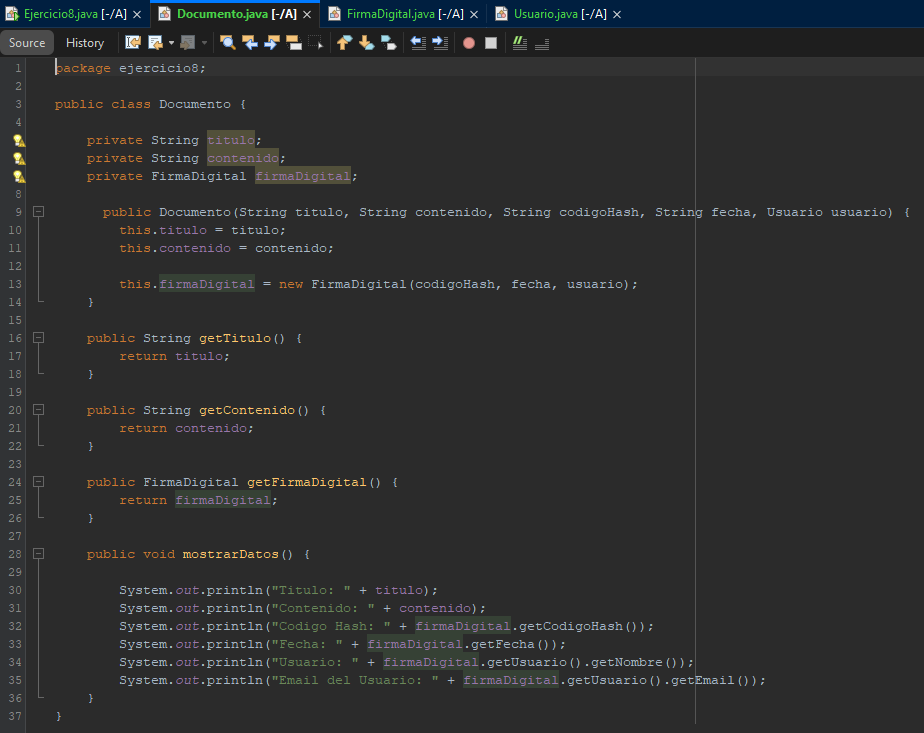
Clases y atributos:

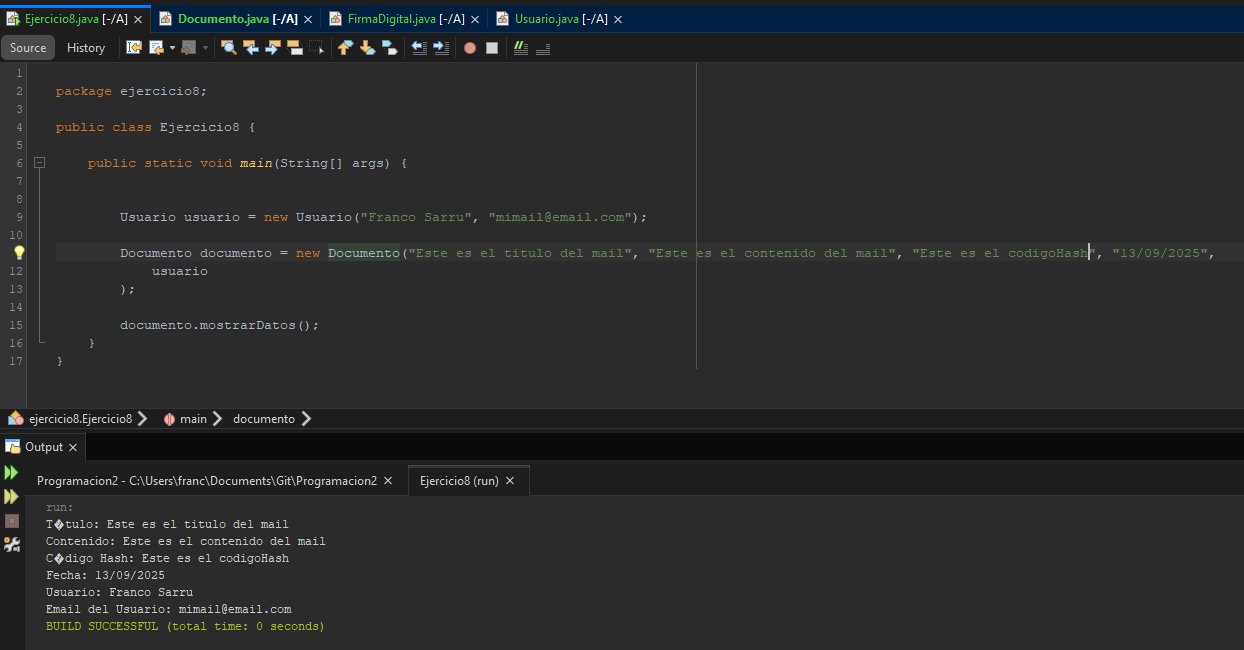
* + 1. Documento: titulo, contenido
    2. FirmaDigital: codigoHash, fecha
    3. Usuario: nombre, email







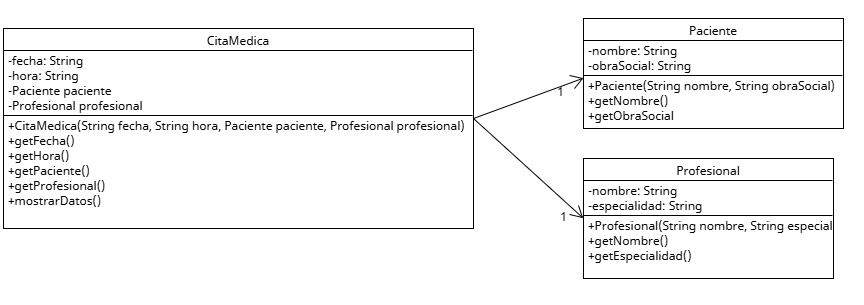


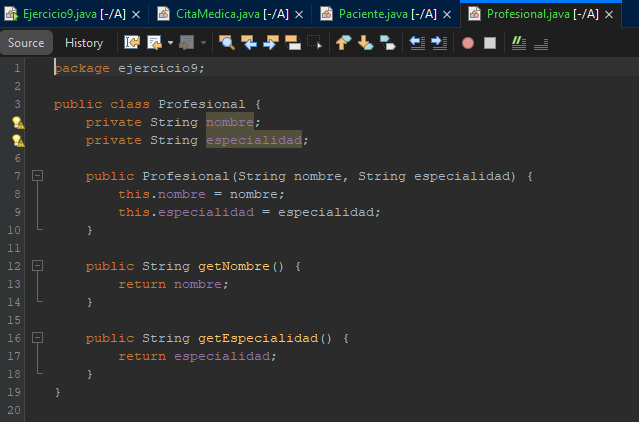


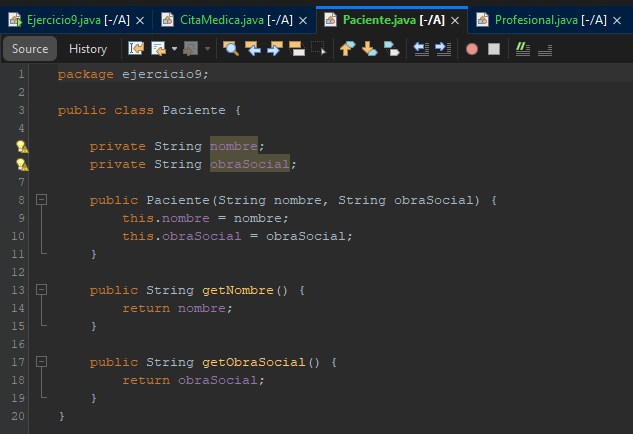
1. CitaMédica - Paciente - Profesional
   1. Asociación unidireccional: **CitaMédica → Paciente,**
   2. Asociación unidirecciona: **CitaMédica → Profesional**

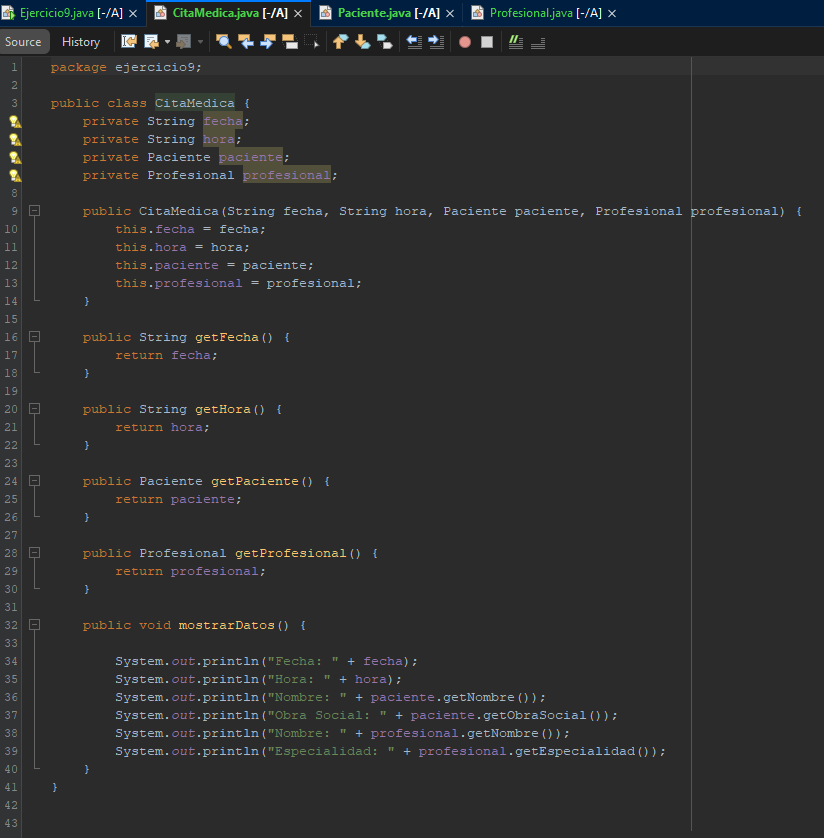
Clases y atributos:

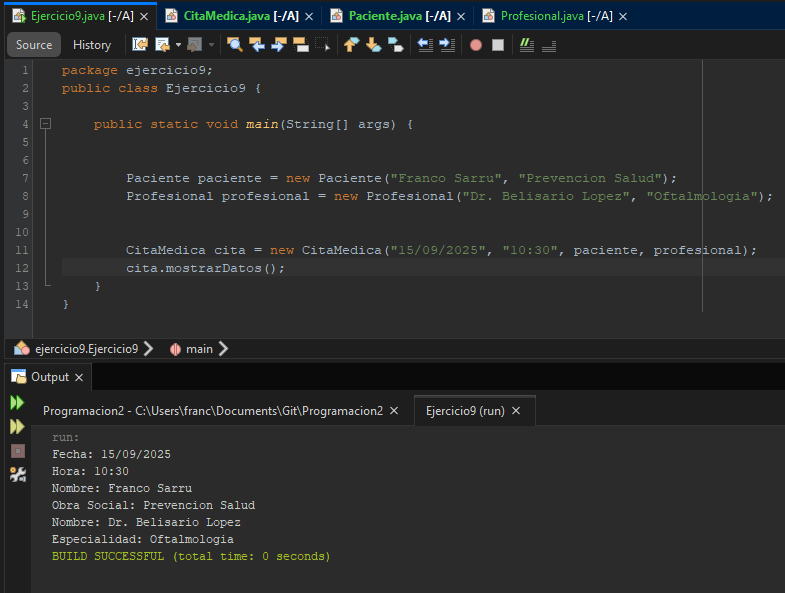
* + 1. CitaMédica: fecha, hora
    2. Paciente: nombre, obraSocial
    3. Profesional: nombre, especialidad







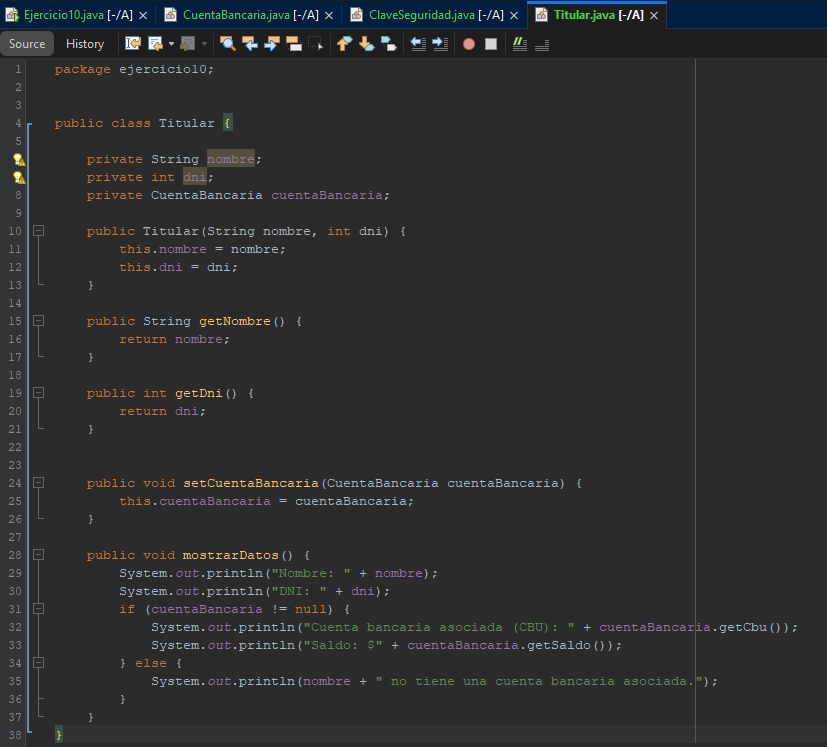
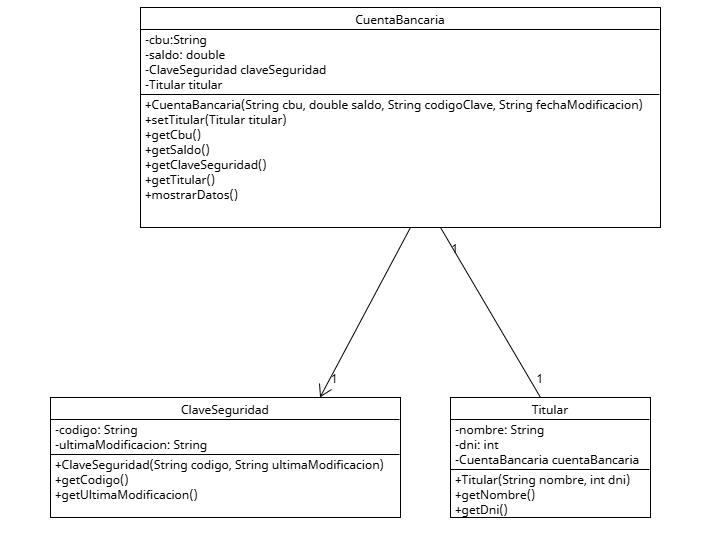


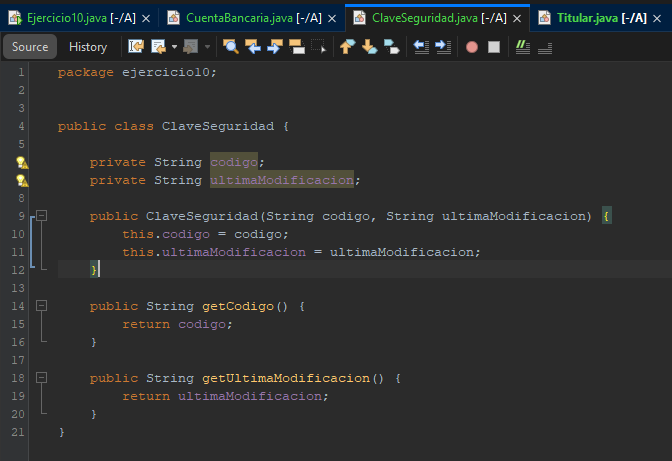


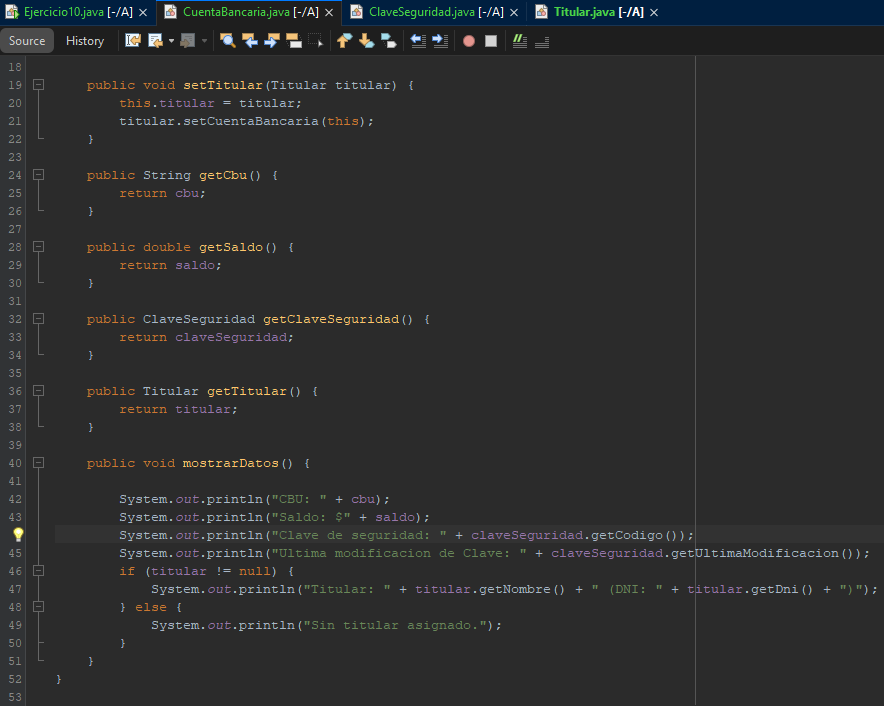
1. CuentaBancaria - ClaveSeguridad - Titular
   1. Composición: **CuentaBancaria → ClaveSeguridad**
   2. Asociación bidireccional: **CuentaBancaria ↔ Titular**

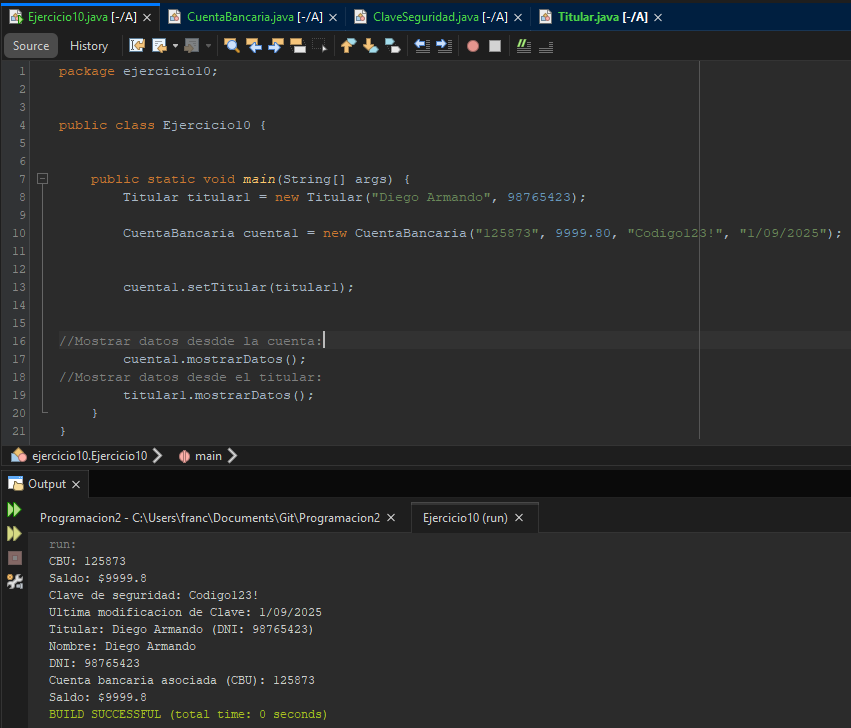
Clases y atributos:

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









## DEPENDENCIA DE USO

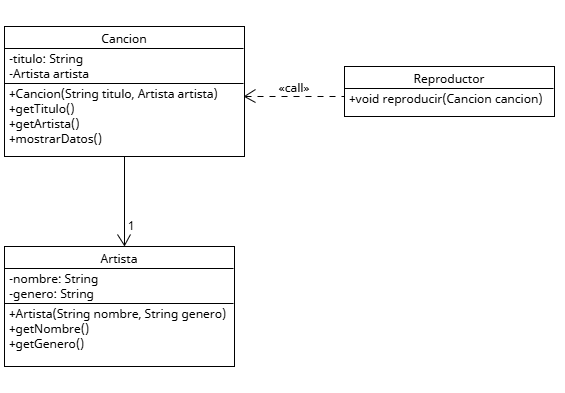
La clase usa otra como **parámetro de un método**, pero **no la guarda como atributo**.

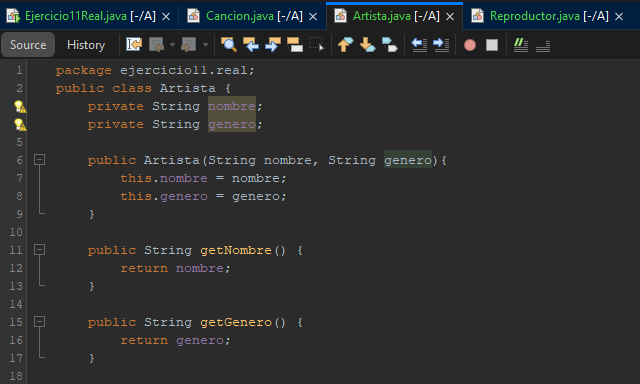
**Ejercicios de Dependencia de Uso**

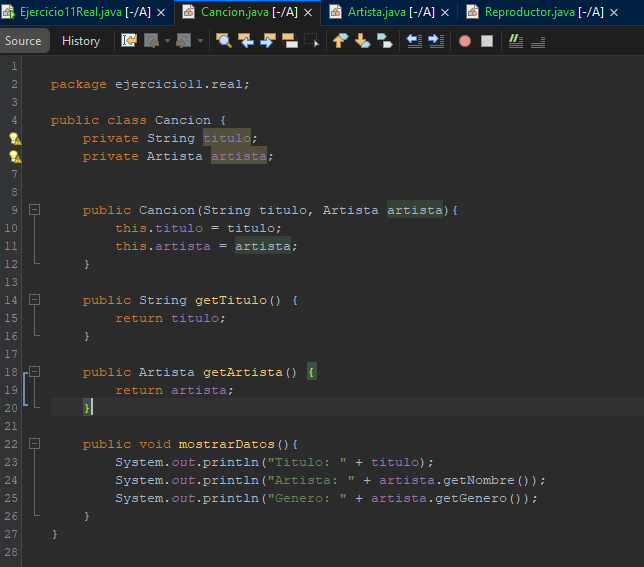
1. Reproductor - Canción - Artista
   1. Asociación unidireccional: **Canción → Artista**
   2. Dependencia de uso: **Reproductor.reproducir(Cancion)**

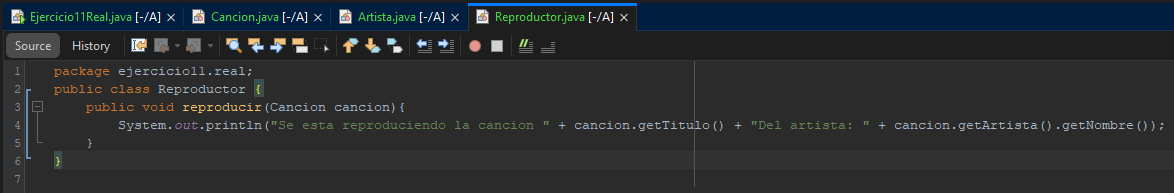
Clases y atributos:

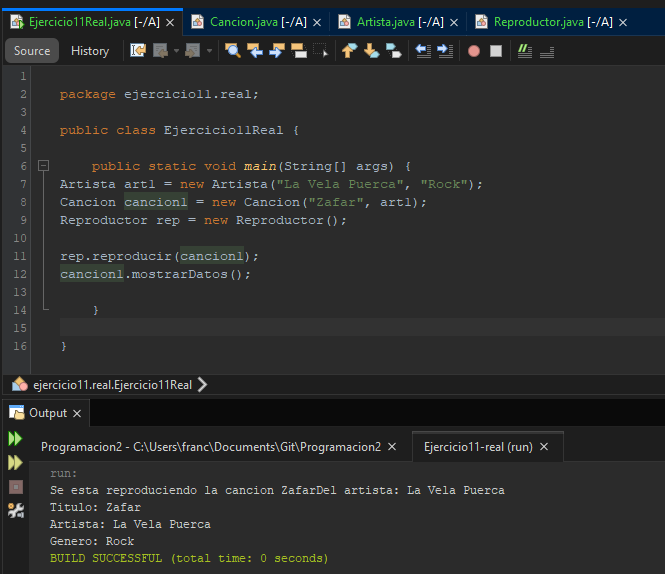
* + 1. Canción: titulo.
    2. Artista: nombre, genero.
    3. Reproductor->método: void reproducir(Cancion cancion)







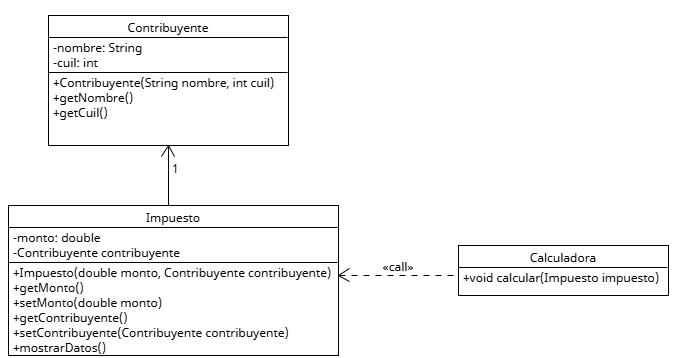


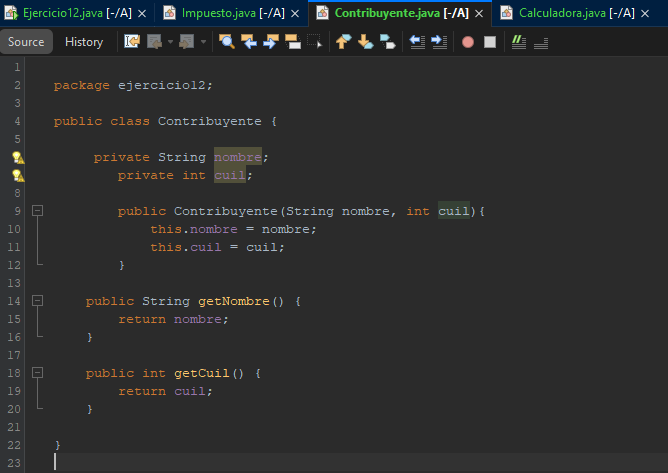


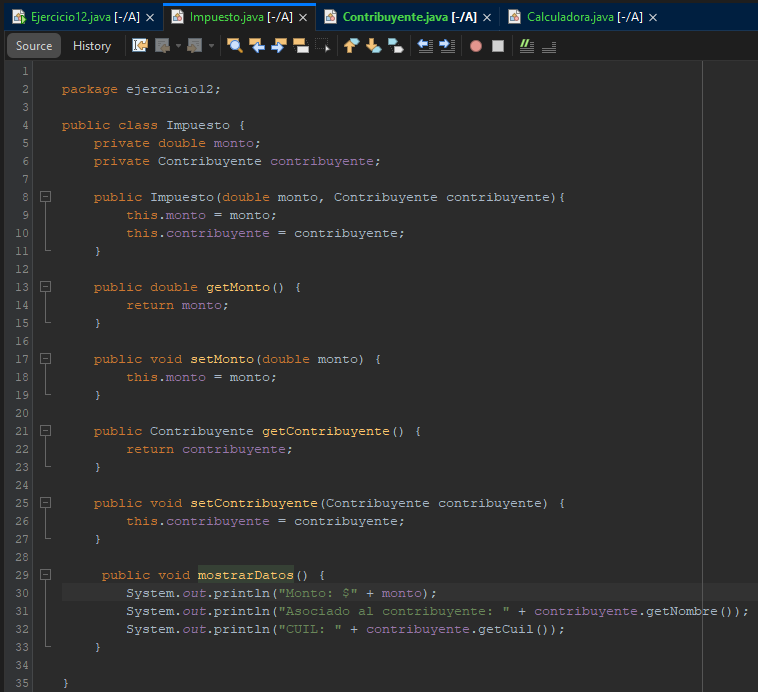
1. Impuesto - Contribuyente - Calculadora
   1. Asociación unidireccional: **Impuesto → Contribuyente**
   2. Dependencia de uso: **Calculadora.calcular(Impuesto)**

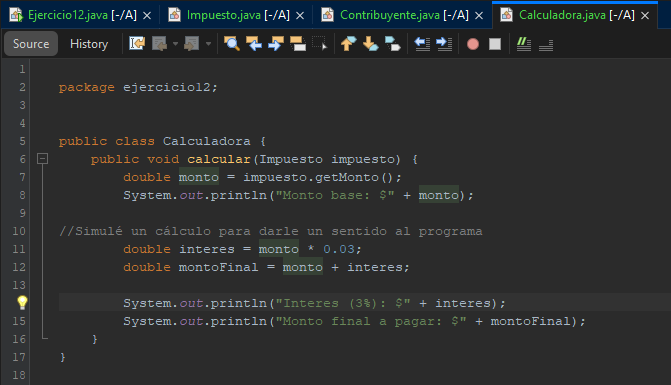
Clases y atributos:

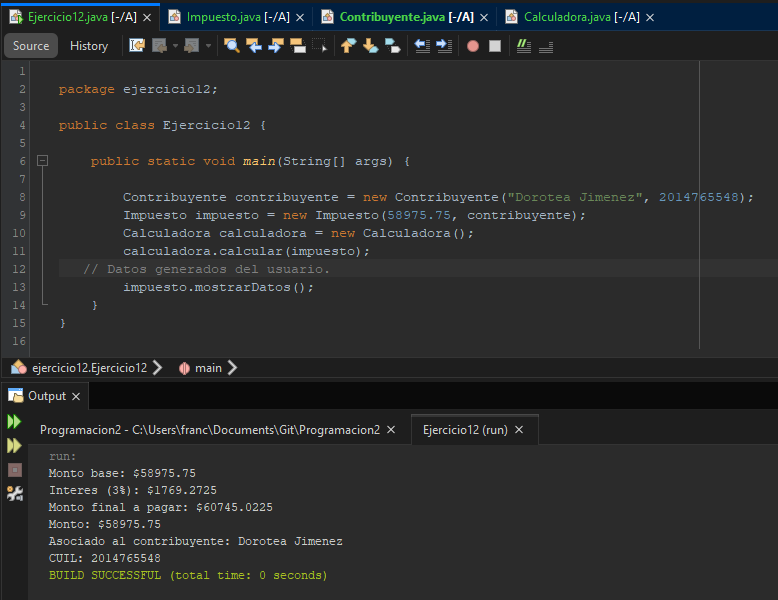
* + 1. Impuesto: monto.
    2. Contribuyente: nombre, cuil.
    3. Calculadora->método: void calcular(Impuesto impuesto)











# DEPENDENCIA DE CREACIÓN

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

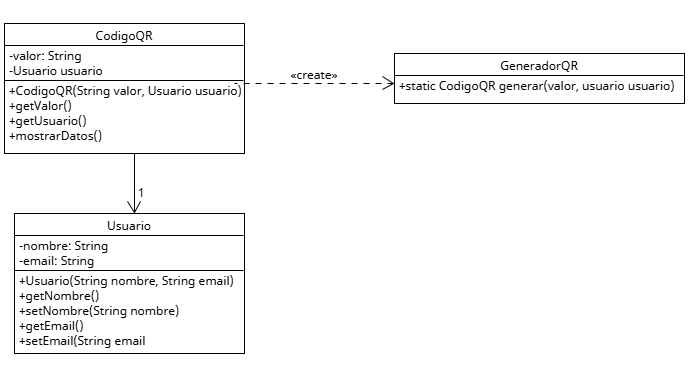
**Ejercicios de Dependencia de Creación**

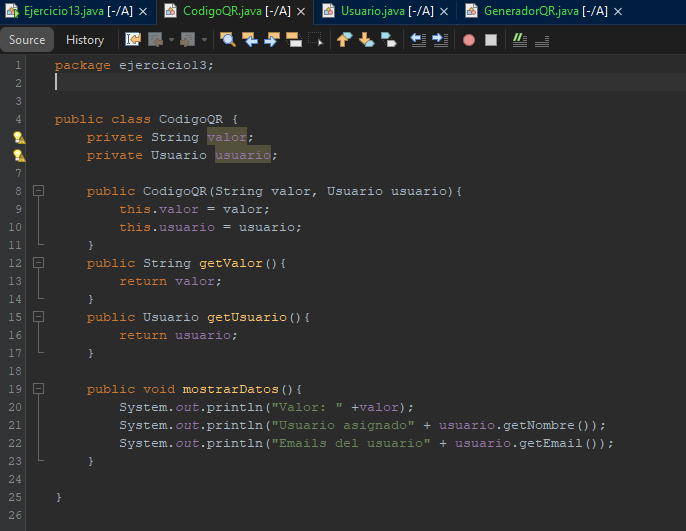
1. GeneradorQR - Usuario - CódigoQR
   1. Asociación unidireccional: **CódigoQR → Usuario**
   2. Dependencia de creación: **GeneradorQR.generar(String, Usuario)**

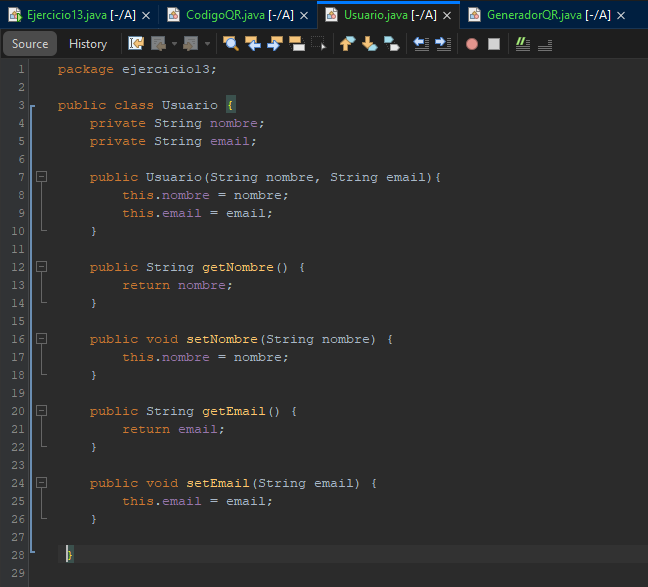
Clases y atributos:

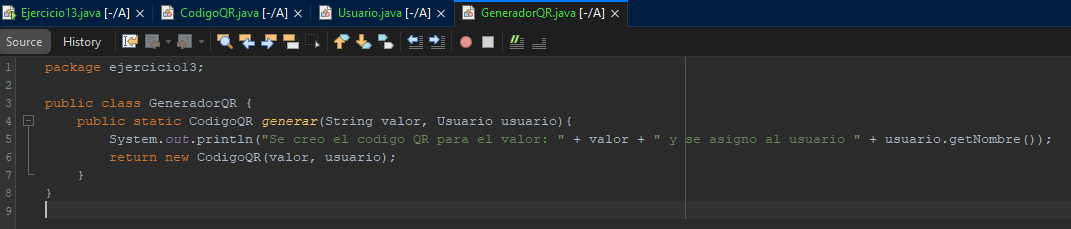
* + 1. CodigoQR: valor.
    2. Usuario: nombre, email.

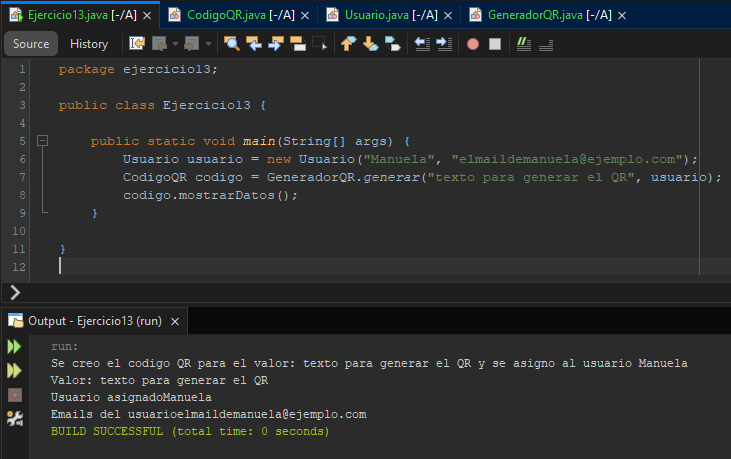
iii. GeneradorQR->método: void generar(String valor, Usuario usuario)

****

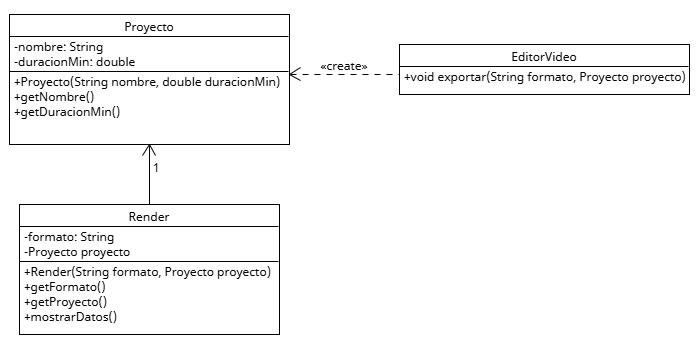
****

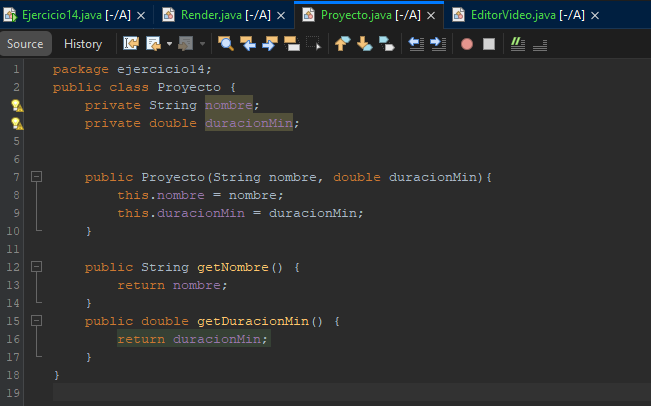
****

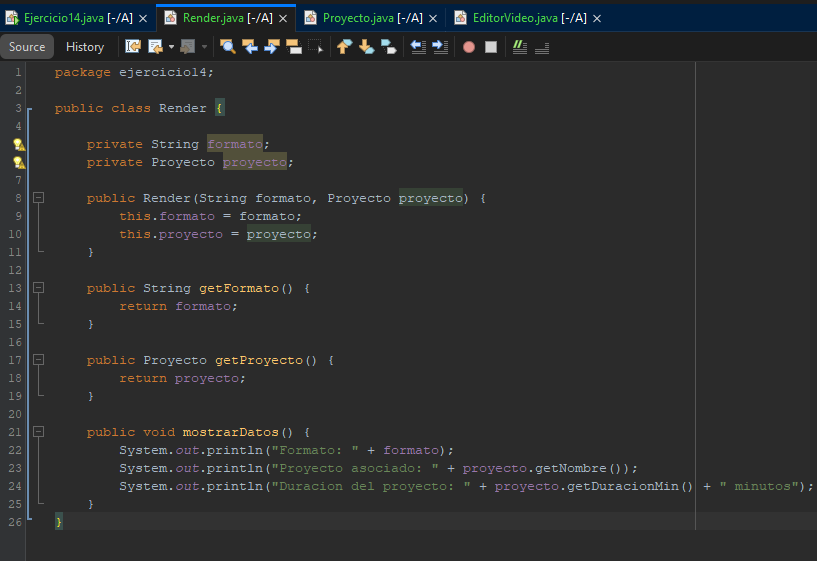
****

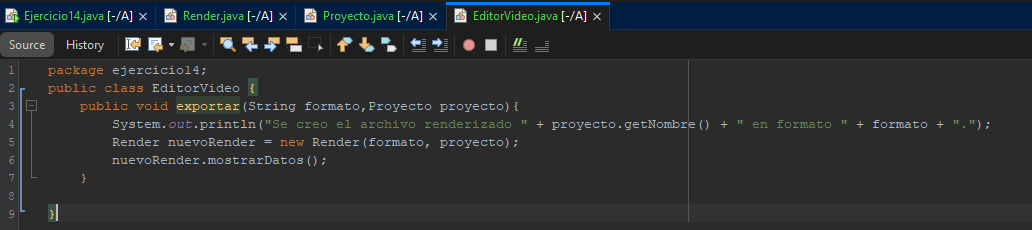
****

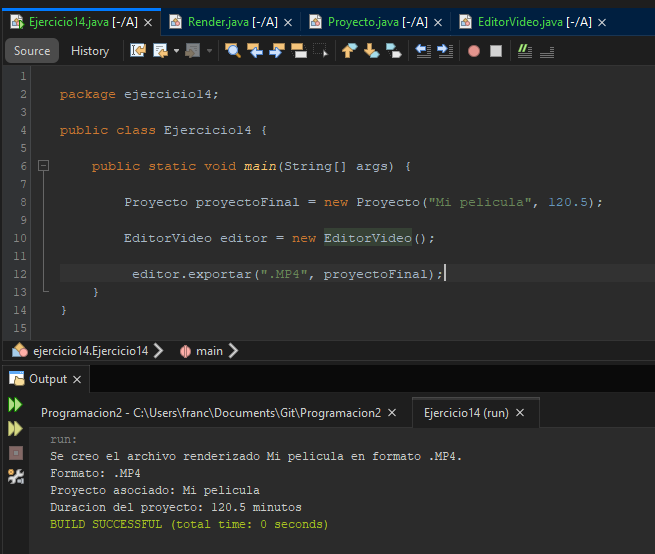
1. EditorVideo - Proyecto - Render
   1. Asociación unidireccional: Render → Proyecto
   2. Dependencia de creación: **EditorVideo.exportar(String, Proyecto)**
   3. Clases y atributos:
      1. Render: formato.
      2. Proyecto: nombre, duracionMin.
      3. iii. EditorVideo->método: void exportar(String formato, Proyecto proyecto)











## CONCLUSIONES ESPERADAS

* Diferenciar claramente los tipos de relaciones entre clases (asociación, agregación, composición).
* Representar las relaciones con la dirección adecuada en diagramas UML. ● Comprender e implementar dependencias de uso y de creación.
* Aplicar relaciones 1 a 1 en el diseño e implementación de clases en Java.
* Reforzar el análisis de modelos orientados a objetos y la capacidad de abstracción.