ADSI – TEMA 5

Análisis

Índice

- Transformación del Modelo del Dominio
  - En una Base de Datos
  - En un Diagrama de Clases
- Diagrama de comunicación
- Arquitectura

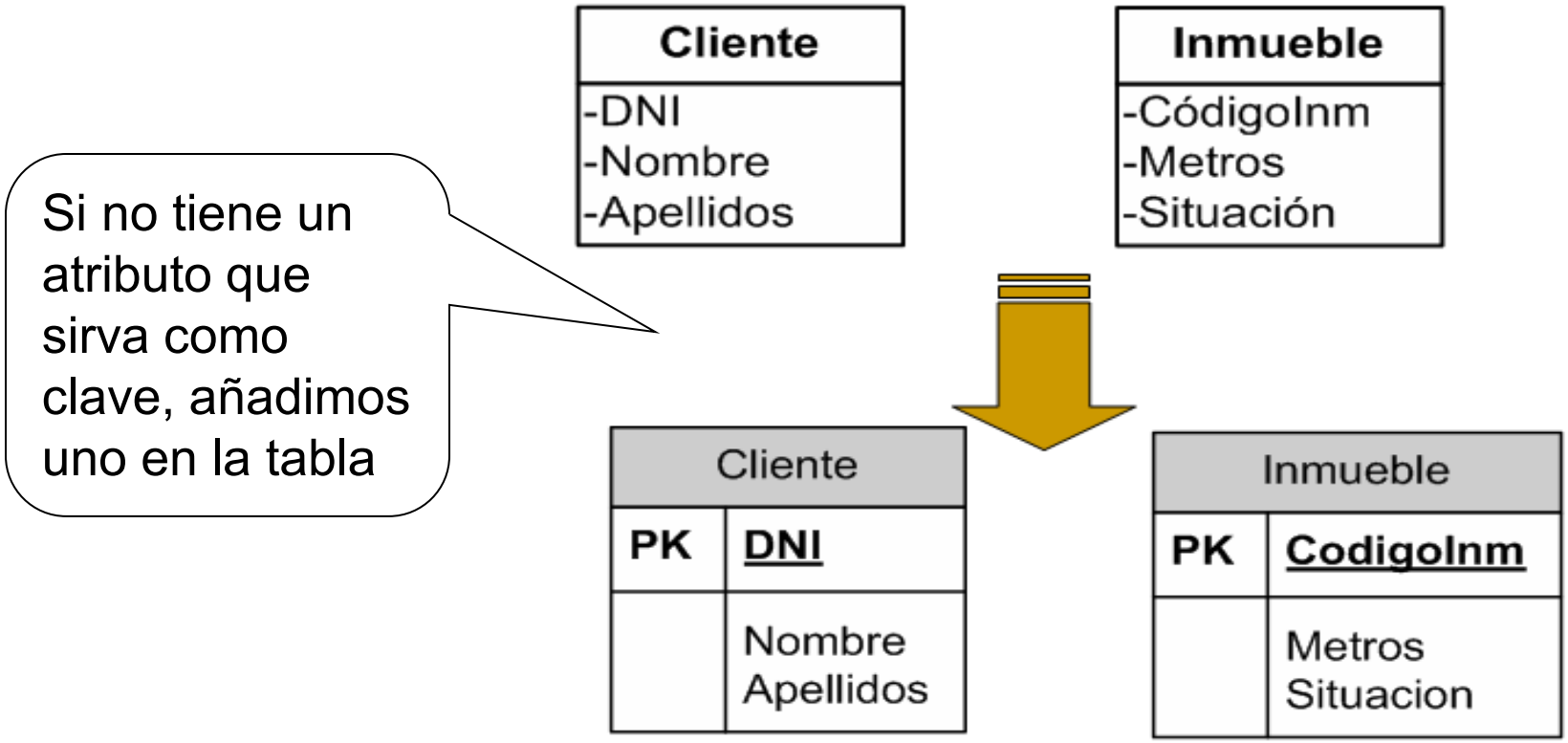
Transformación del MD

- Tenemos que decidir qué tipo de sistema vamos a desarrollar
  - Usando una Base de Datos relacional
    - Tendremos que definir las tablas, sus claves primarias, sus claves extranjeras, etc.
  - Orientado a Objetos
    - Tendremos que definir las clases, sus atributos, etc.
  - Ambas
    - Tendremos que definir todo lo anterior

Es lo más común

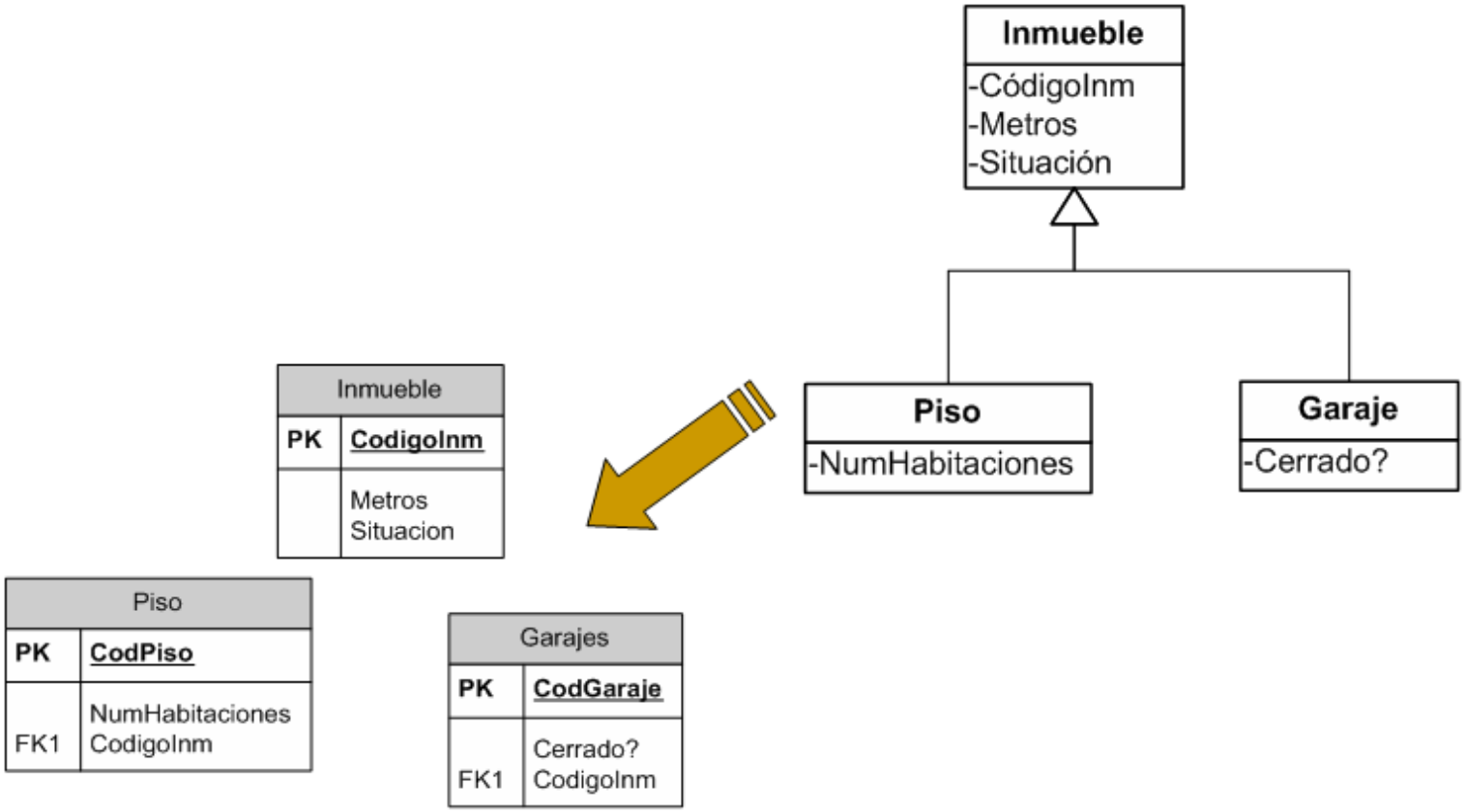
Transformación del MD en BD

- Cada entidad se transforma en una tabla con su propia clave primaria



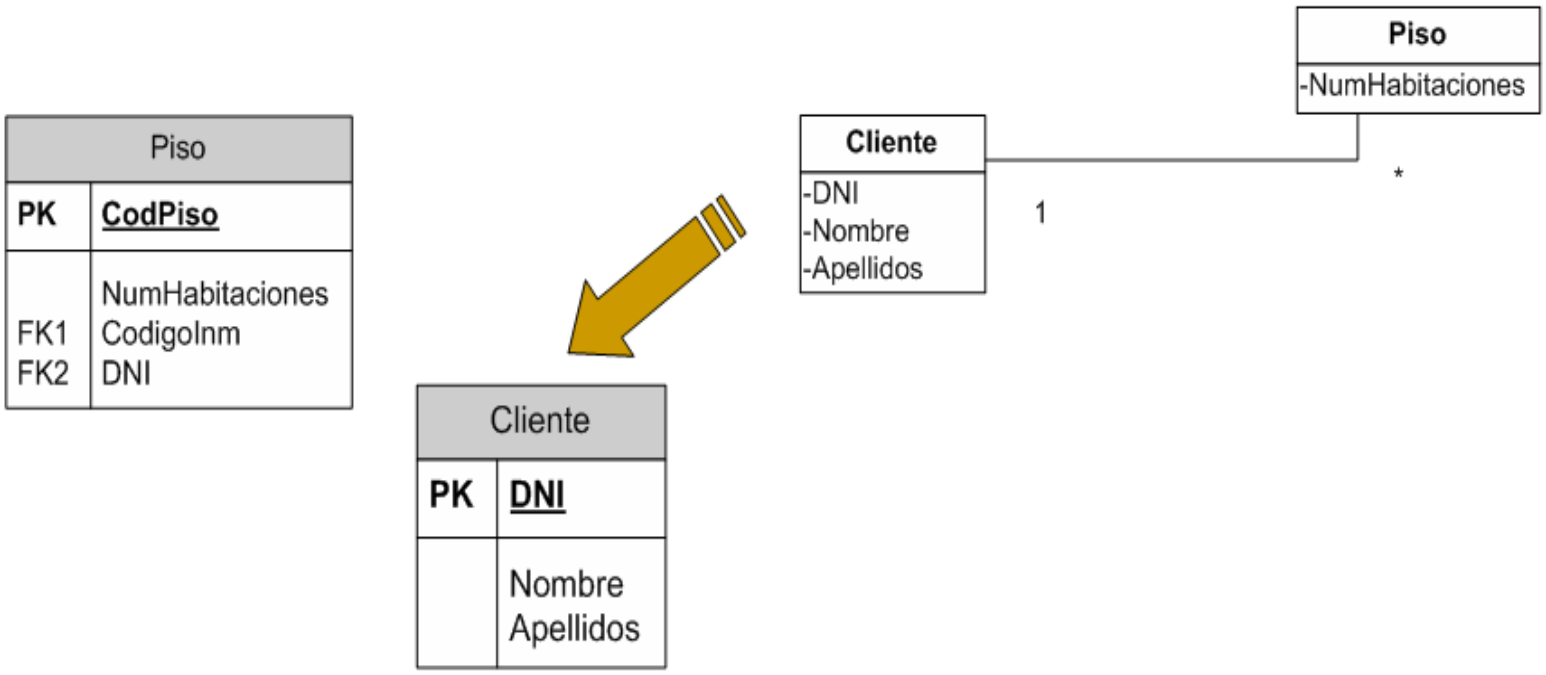
Transformación del MD en BD

- Si hay herencia, la clave primaria del padre pasa a los hijos como clave extranjera



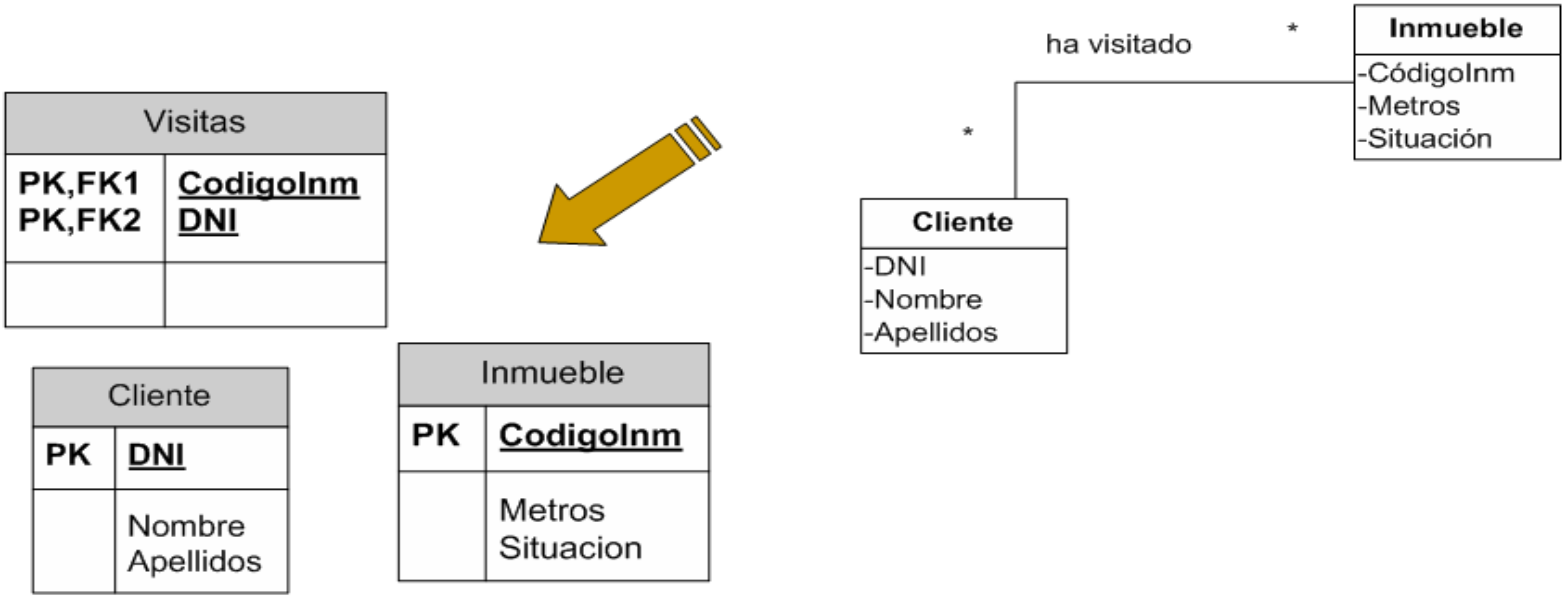
Transformación del MD en BD

- Una relación binaria.. Depende de la cardinalidad de la relación
  - 1..* ► La clave de la parte 1, pasa a la tabla de la parte * como clave extranjera



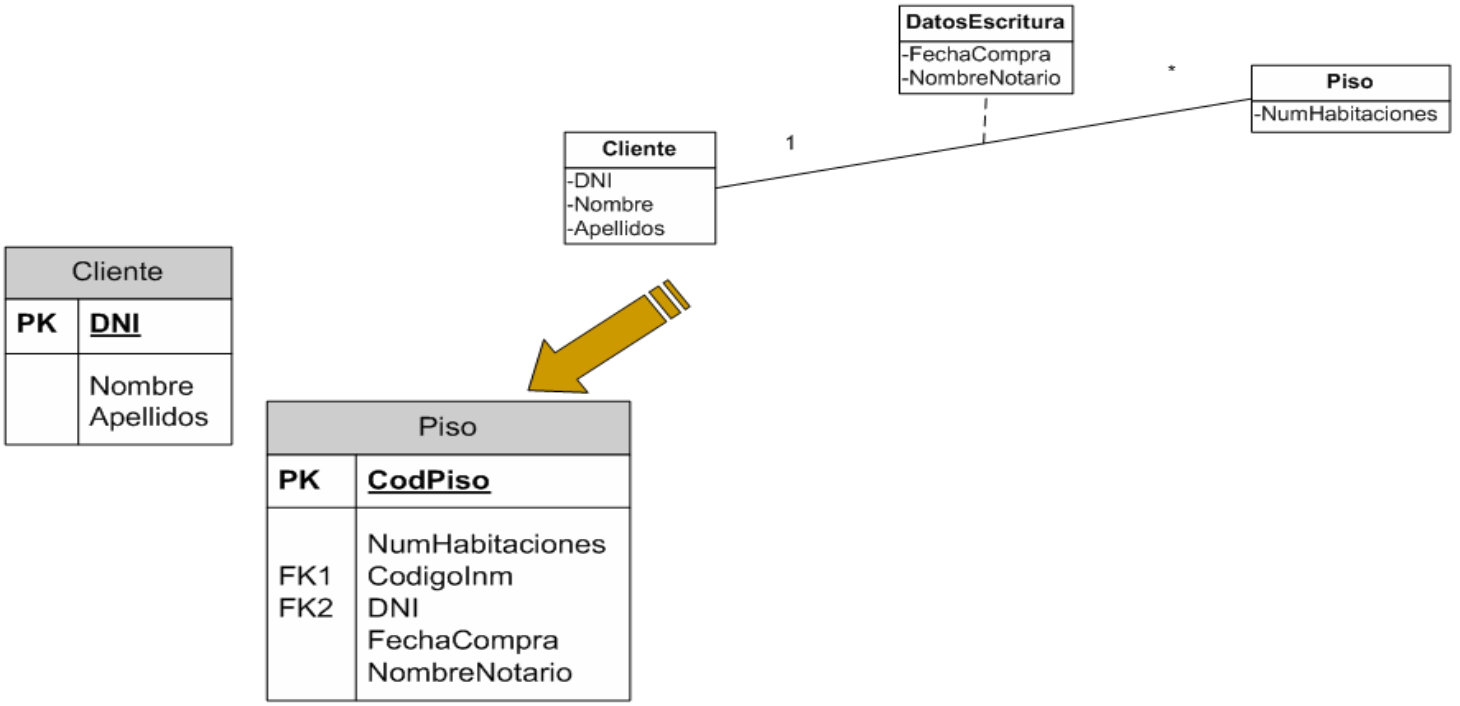
Transformación del MD en BD

- Una relación binaria.. Depende de la cardinalidad de la relación
  - *..* ► Se crea una tabla intermedia cuya clave está formada por la combinación de las claves primarias de las tablas de la relación



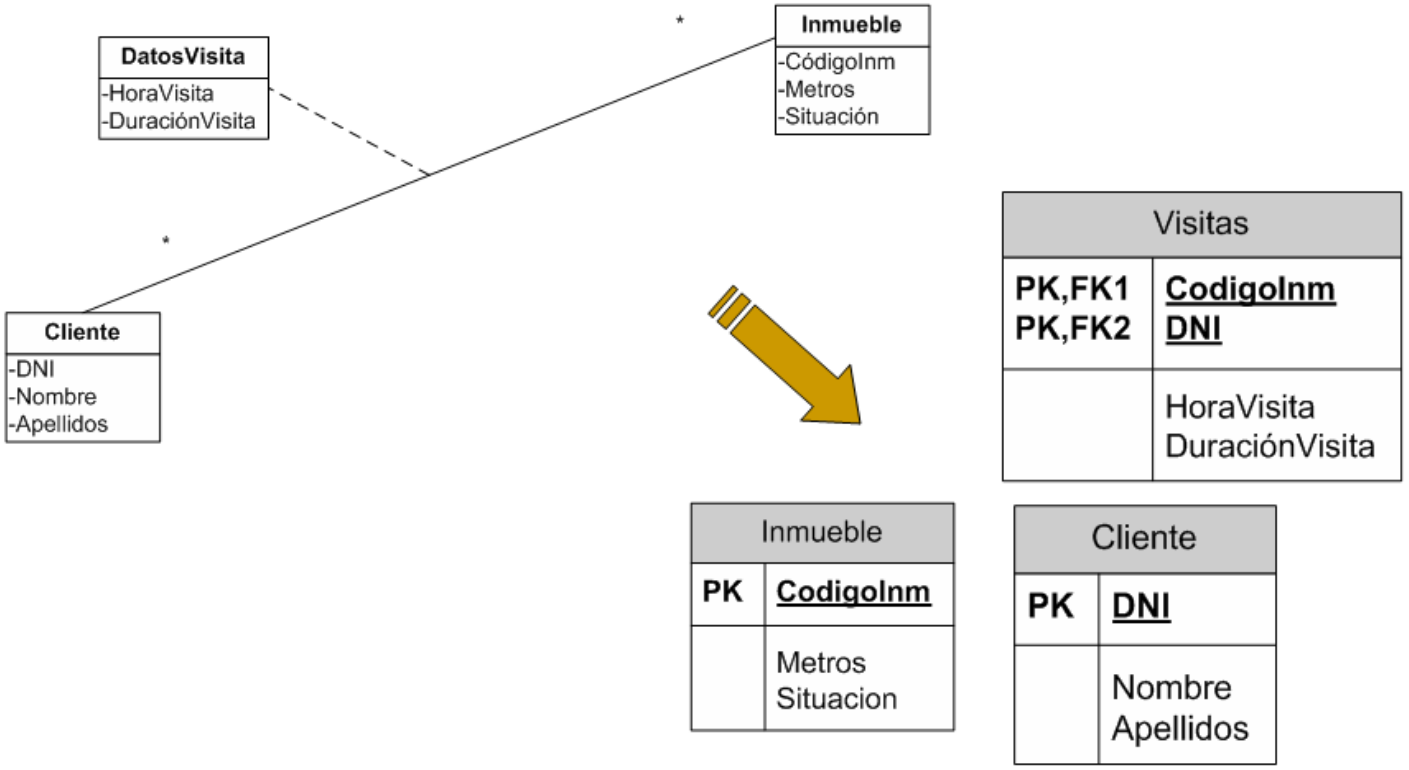
Transformación del MD en BD

- Si hay una entidad asociación, sus atributos "viajan" con las claves



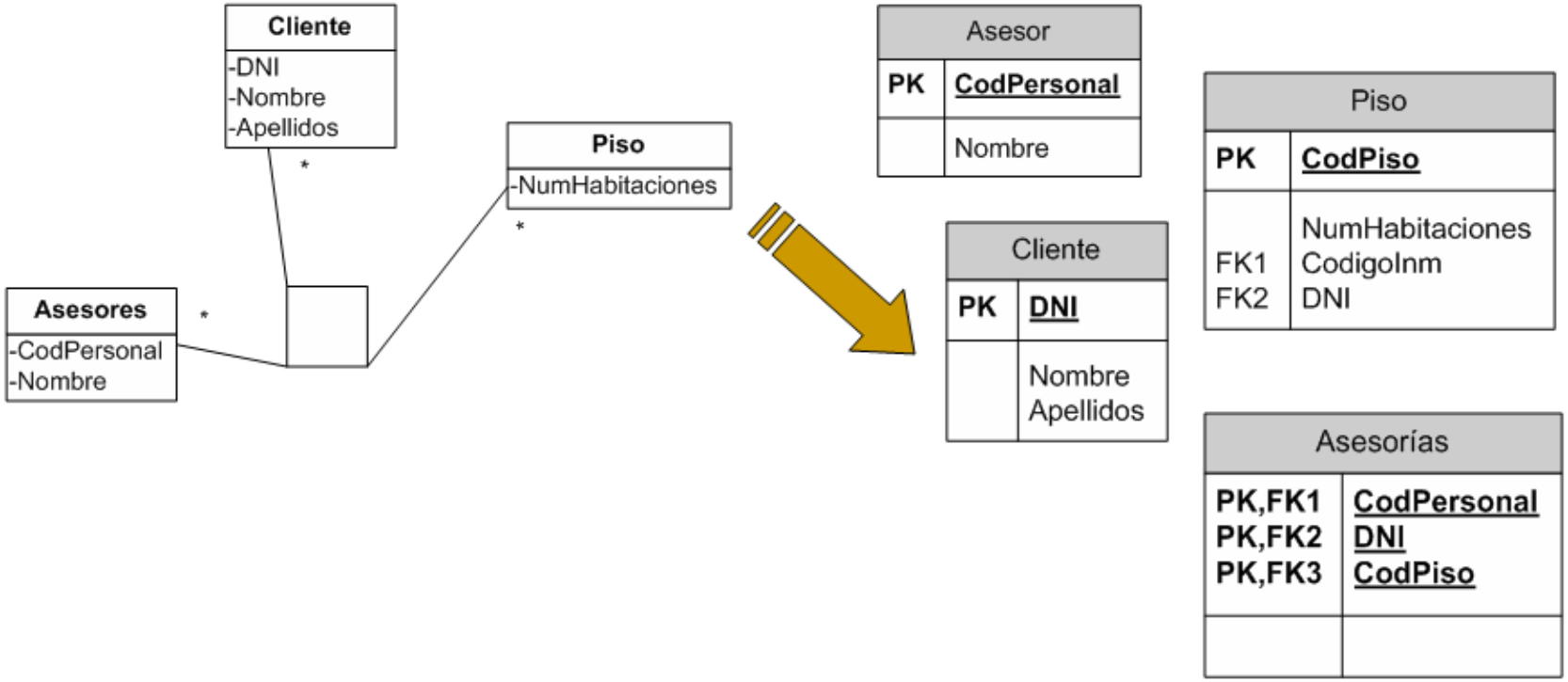
Transformación del MD en BD

- Si hay una entidad asociación, sus atributos "viajan" con las claves



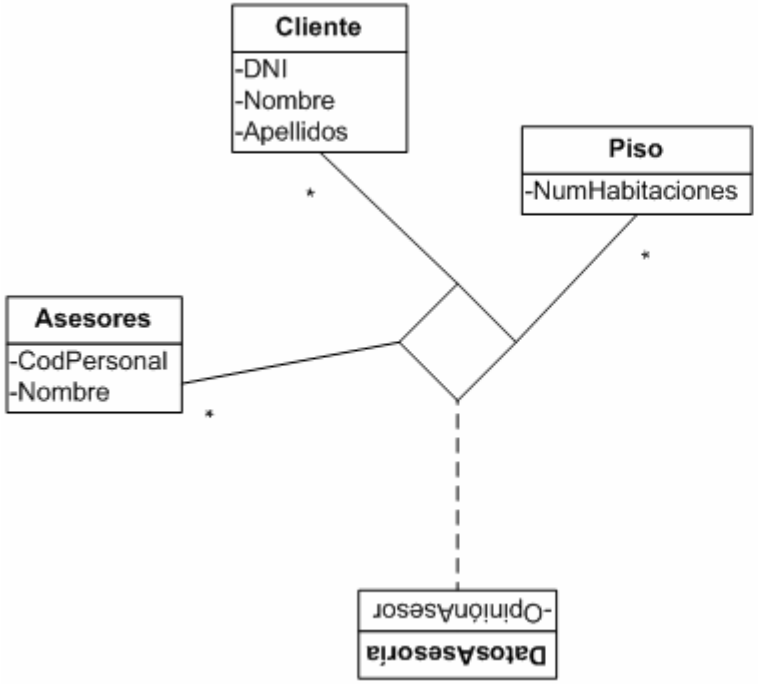
Transformación del MD en BD

- Una relación múltiple se convierte en una tabla cuya clave principal es la combinación de las claves primarias de las entidades de la



Transformación del MD en BD

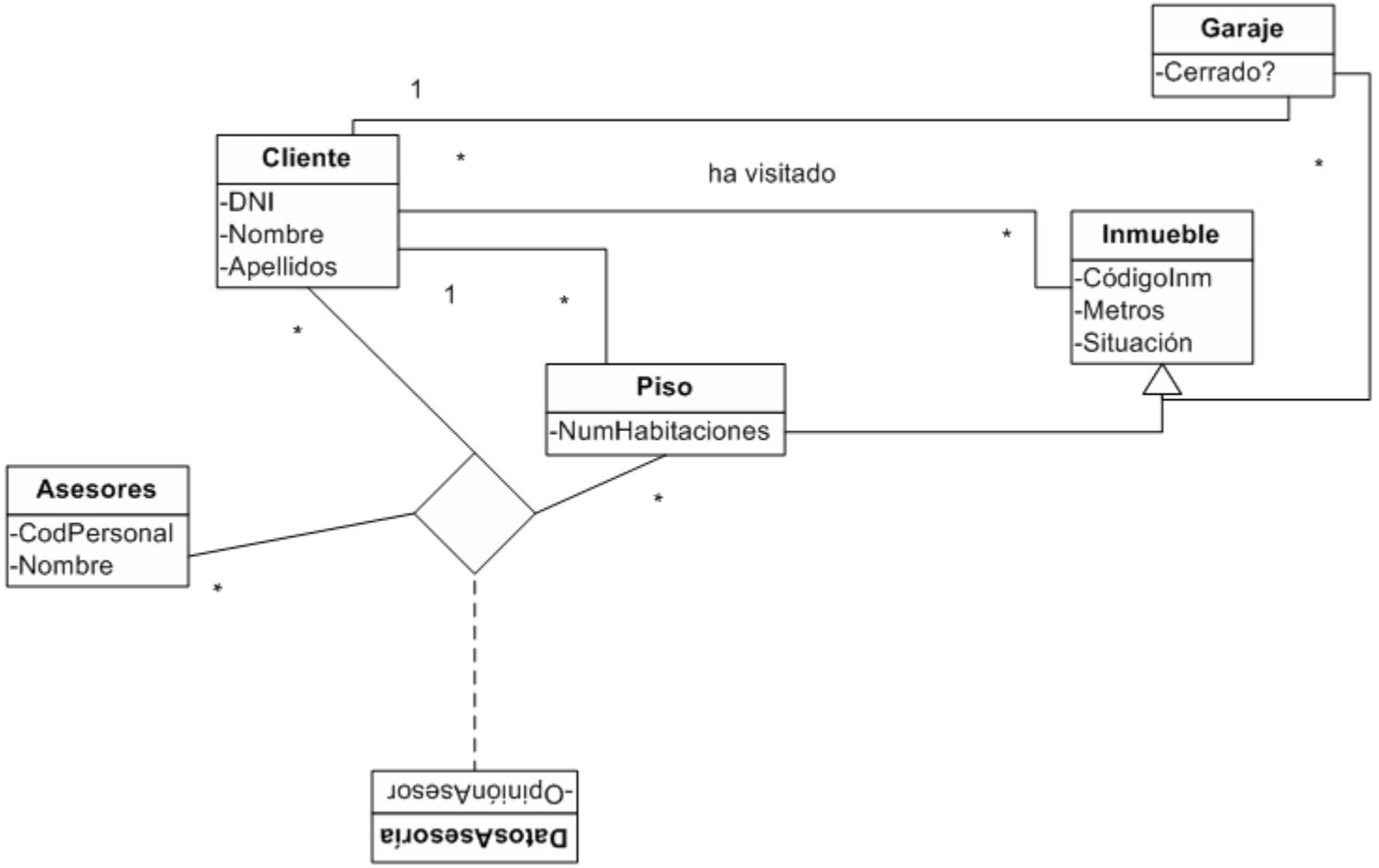
- Si en la relación múltiple hay atributos, estos van a la nueva tabla



Asesorías	
PK,FK1	**CodPersonal**
PK,FK2	**DNI**
PK,FK3	**CodPiso**
	OpiniónAsesor

Transformación del MD en BD

- Ejemplo

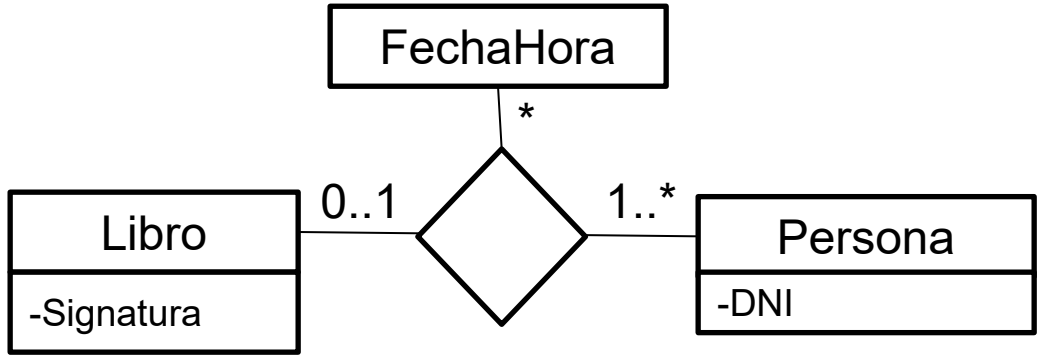


Transformación del MD en BD

- Excepción
  - Las entidades que representan tipos de datos que existen en el SGBD a utilizar, no se transforman en tablas
    - Fecha
    - FechaHora
    - Año (entero)
    - …
  - Pero sí se tienen en cuenta al hacer la transformación

Transformación del MD en BD

- Excepción



Libro	
PK	**Signatura**

Persona	
PK	**DNI**

Lectura	
PK,FK1	**DNI**
PK, FK2	**Signatura**
PK	**FechaHora**

No ponemos una tabla FechaHora, porque el SGBD a usar (por ejemplo, MySQL) tiene un tipo de datos DATETIME, pero sí que ponemos un atributo FechaHora que formará parte de la clave primaria

Transformación del MD en BD

- Ejemplo

Inmueble	
PK	**CodigoInm**
	Metros Situacion

Garajes	
PK	**CodGaraje**
FK1 FK2	Cerrado? CodigoInm DNI

Piso	
PK	**CodPiso**
FK1 FK2	NumHabitaciones CodigoInm DNI

Asesor	
PK	**CodPersonal**
	Nombre

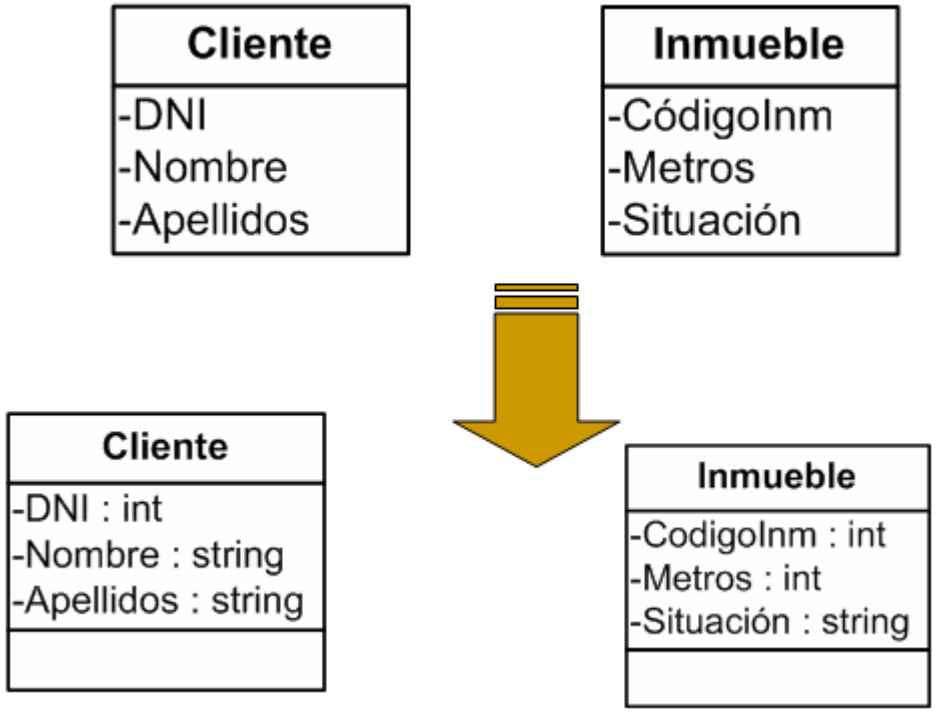
Cliente	
PK	**DNI**
	Nombre Apellidos

Visitas	
PK,FK1 **PK,FK2**	**CodigoInm** **DNI**

Asesorías	
PK,FK1 **PK,FK2** **PK,FK3**	**CodPersonal** **DNI** **CodPiso**
	OpiniónAsesor

Transformación del MD en DC

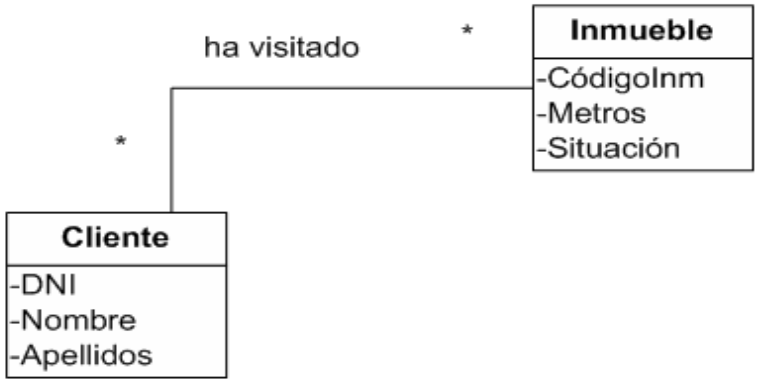
- Cada entidad será una clase con sus atributos y el tipo de datos correspondiente



Transformación del MD en DC

- Las relaciones binarias hay 3 formas de implementarlas, elegir entre una u otra depende la funcionalidad que tenga el sistema

Las 3 son igual de correctas



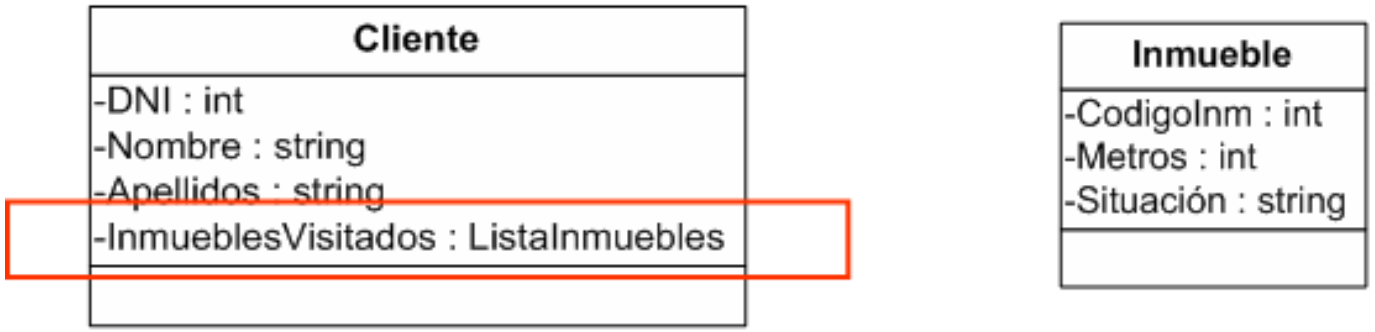
  - Implementar la relación en un lado (Cliente)
  - Implementar la relación en el otro lado (Inmueble)
  - Implementar la relación en los dos lados

Transformación del MD en DC

- Las relaciones, se implementarán en forma de Atributos o de Listas de Elementos
- Una vez decidido en qué lado se implementa (puede ser en los dos)
  - Si la cardinalidad de la relación EN EL LADO CONTRARIO es 1, lo implementaremos mediante un **atributo**
  - Si la cardinalidad de la relación EN EL LADO CONTRARIO es *, lo implementaremos mediante una **lista de elementos**

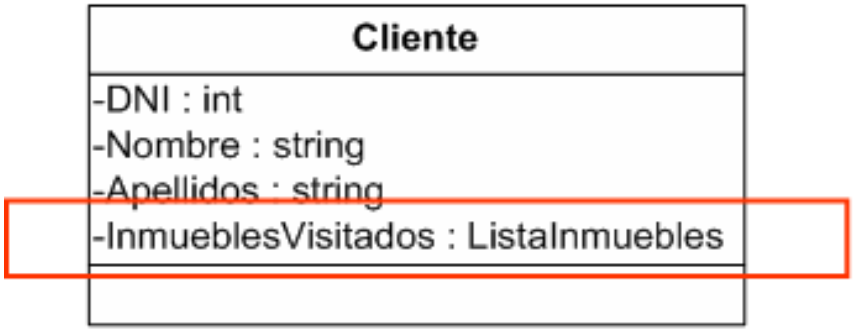
Transformación del MD en DC

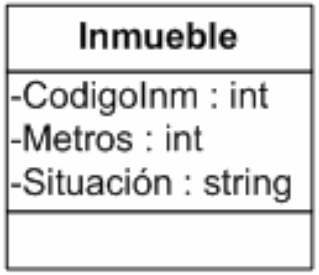
- Implementar la relación en un lado (Cliente)
  - Como un cliente ha visitado * Inmuebles, se implementa mediante una lista



Transformación del MD en DC

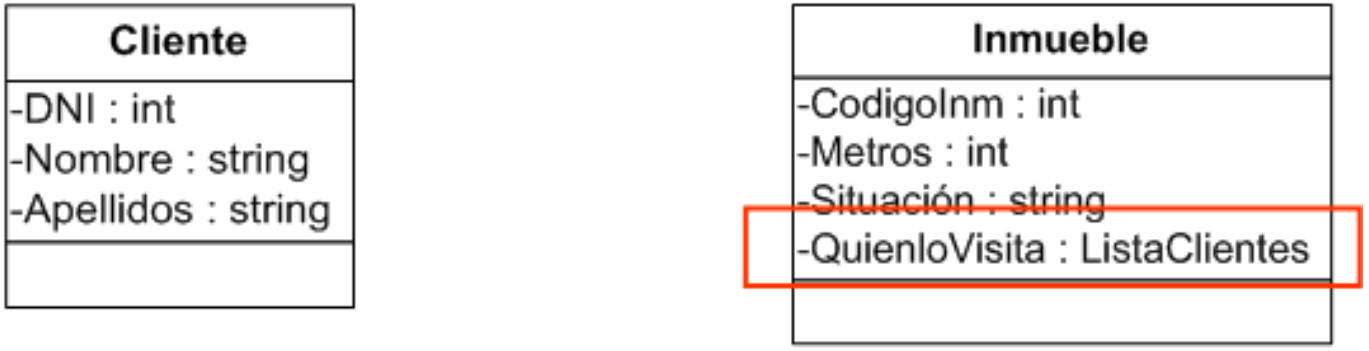
- Ventajas:
  - Dado un cliente se puede ver fácilmente qué inmuebles ha visitado.
  - Añadir una visita, consiste en añadir a la lista, la referencia del inmueble
- Desventajas:
  - Dado un inmueble no se puede ver fácilmente quién lo ha visitado





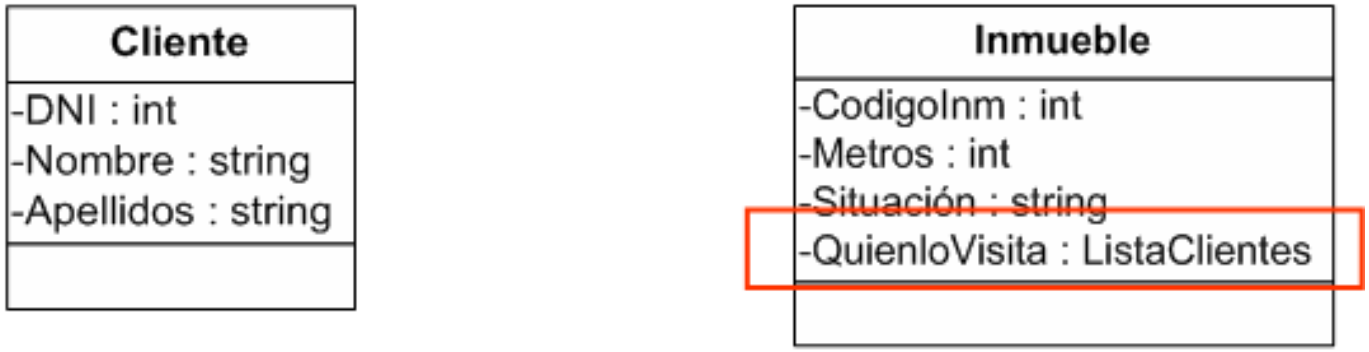
Transformación del MD en DC

- Implementar la relación en el otro lado (Inmueble)
  - Como un inmueble ha podido ser visitado por * Clientes, se implementa mediante una lista



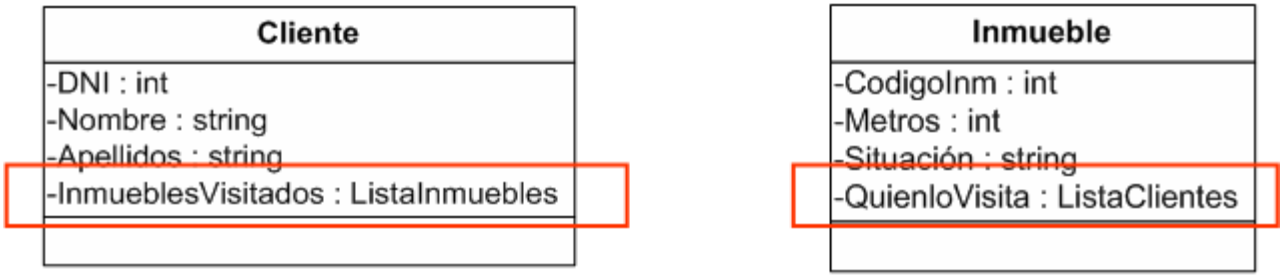
Transformación del MD en DC

- Ventajas:
  - Dado un Inmueble se puede ver fácilmente quién lo ha visitado.
  - Añadir una visita, consiste en añadir a la lista, la referencia al Cliente.
- Desventajas:
  - Dado un cliente no se puede ver fácilmente qué Inmuebles ha visitado



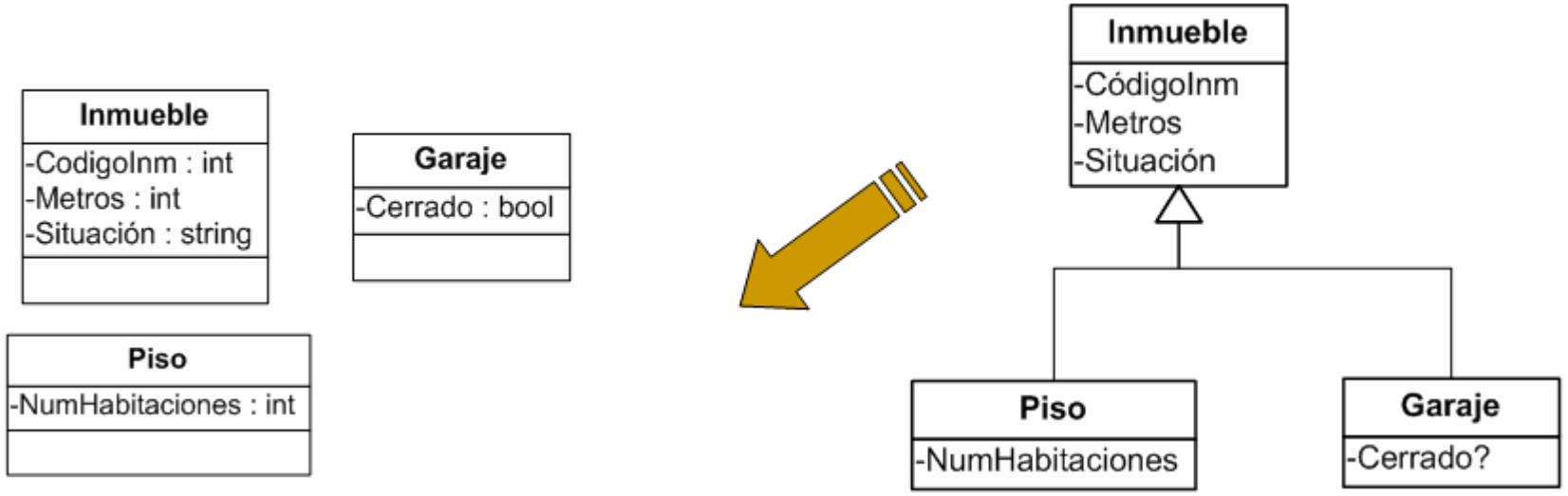
Transformación del MD en DC

- Implementar la relación en ambos lados
  - Como las dos cardinalidades son *, se implementa mediante dos listas
  - Ventajas:
    - Puedo saber fácilmente toda la información.
  - Desventajas:
    - Añadir una visita supone actualizar las dos listas.



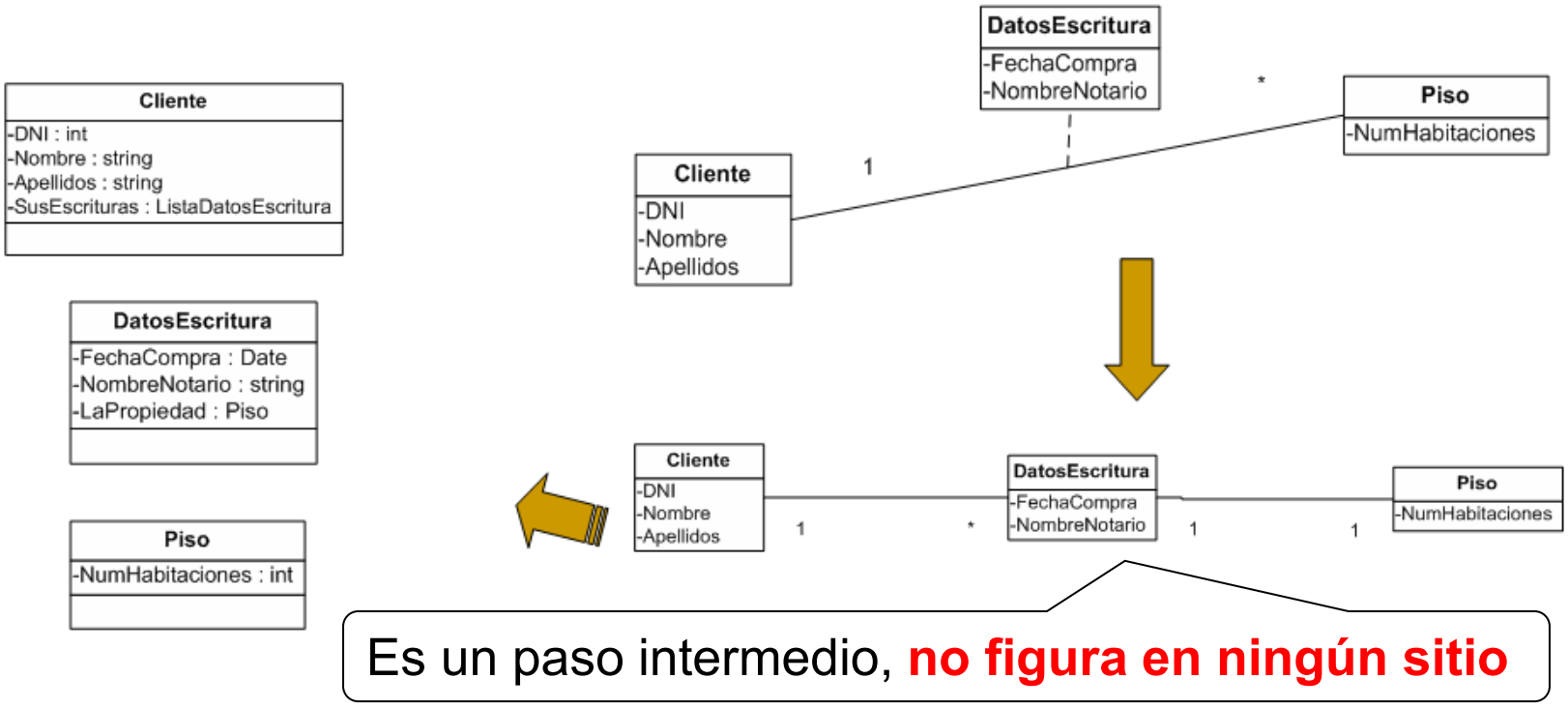
Transformación del MD en DC

- Si hay herencia cada entidad será una clase y desde las clases hijas se accederá a los atributos y operaciones del padre gracias a los mecanismos de la OO



Transformación del MD en DC

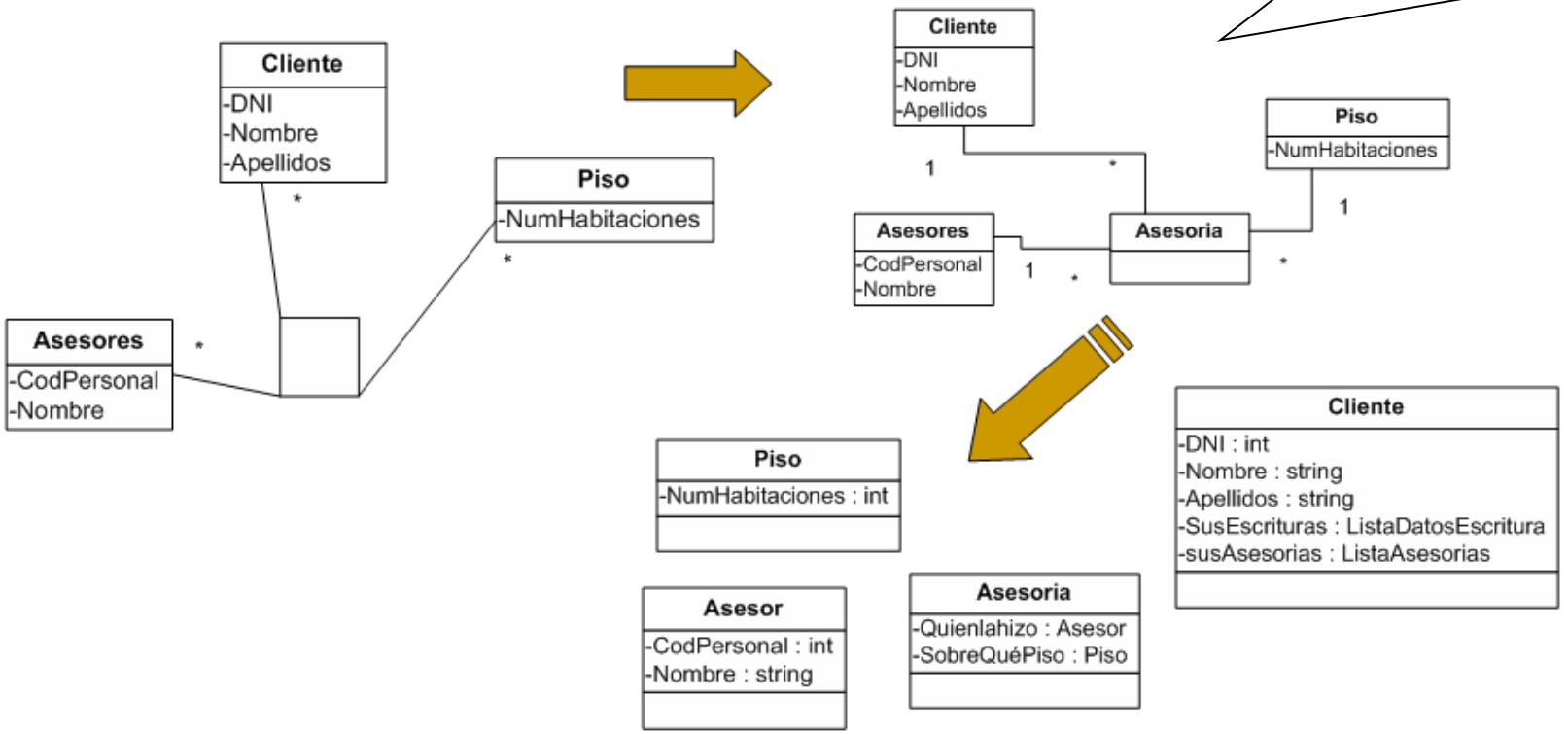
- Si hay una asociación, se trata como si fuera una entidad con sentido de manera independiente



Transformación del MD en DC

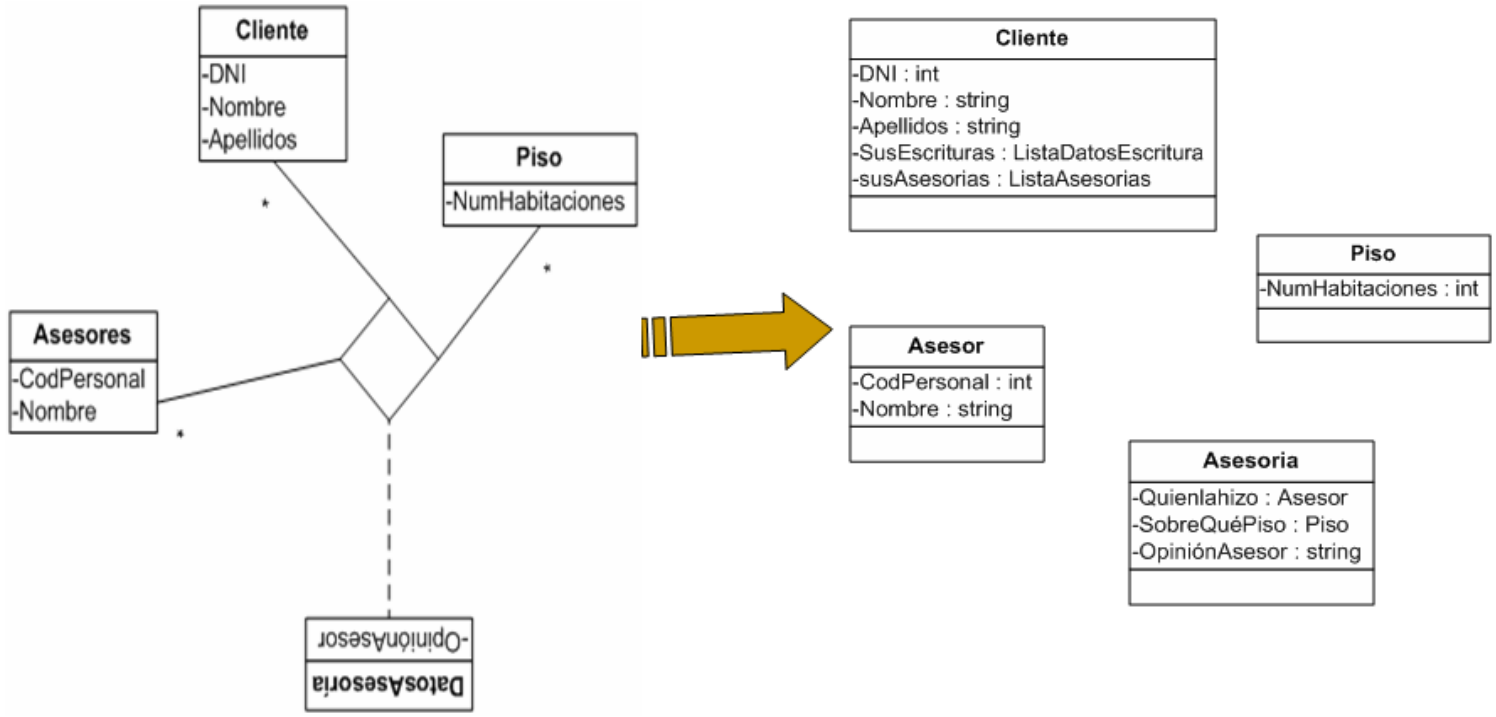
- Una relación múltiple se trata como si fuera una nueva entidad

Es un paso intermedio, **no figura en ningún sitio**



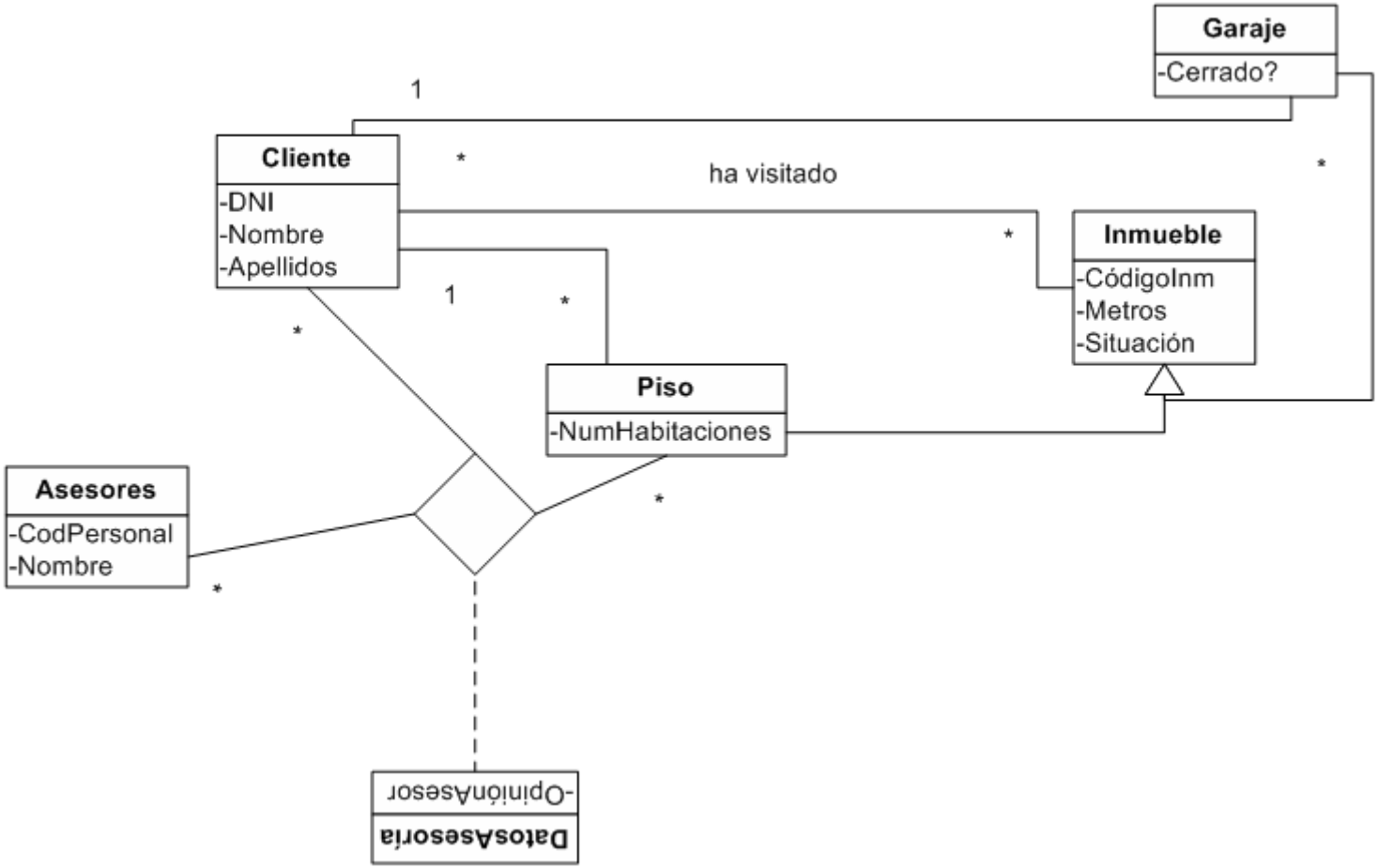
Transformación del MD en DC

- Si en la relación hay atributos, estos van a la nueva clase



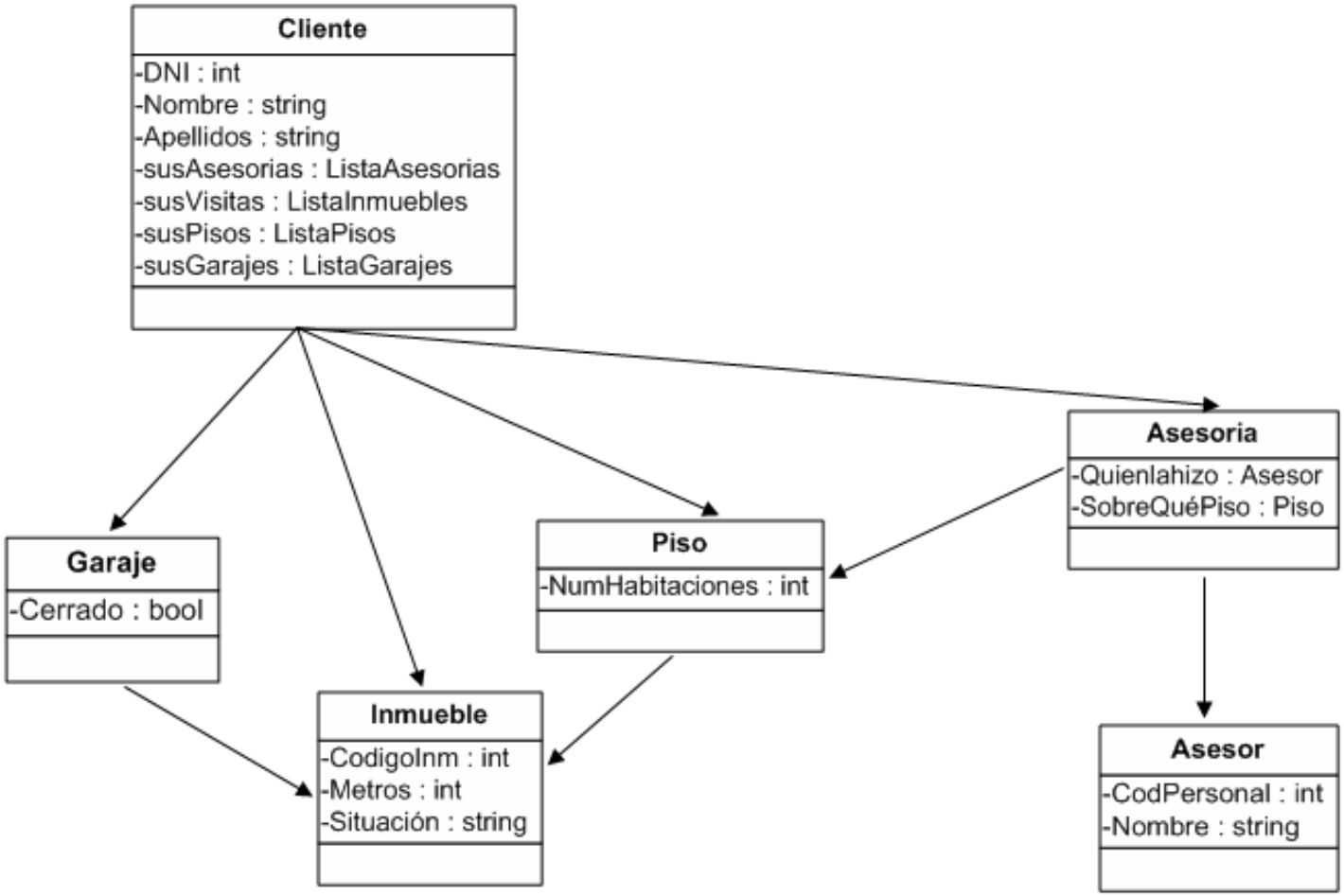
Transformación del MD en DC

- Ejemplo



Transformación del MD en DC

- Ejemplo

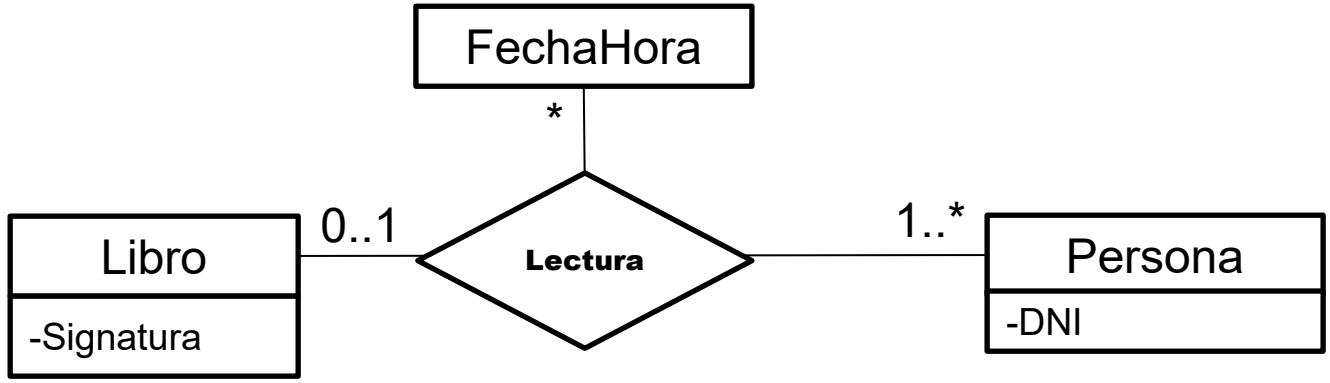


Transformación del MD en DC

- Excepción
  - Las entidades que representan tipos de datos que existan en el lenguaje de programación a utilizar, no se transforman en nuevas clases
    - Fecha
    - FechaHora
    - Año
    - …
  - Pero sí se tienen en cuenta al hacer la transformación

Transformación del MD en DC

- Excepción



Libro
-Signatura: string

Persona
-DNI: string -SusLecturas: ListaLectura

Lectura
-Que: Libro -Cuando: Date

No creamos una clase FechaHora porque existe la clase Date, pero sí que ponemos un atributo que haga referencia a ese tipo de datos

Diagrama de comunicación

- Hasta UML 1.4 llamado Diagrama de colaboración
- Esta Fase está especialmente orientada a su uso con Orientación a Objetos
- Se obtendrán más clases para añadir al diagrama de clases → Controlador
- Se obtendrá qué operaciones se necesitan en cada clase

Diagrama de comunicación

- Hay decidir qué operaciones se definen en cada clase
  - Nombre
  - Resultado
  - Parámetros
  - ….

*Operación que devuelve los datos personales contenidos en una instancia de la clase **Persona***

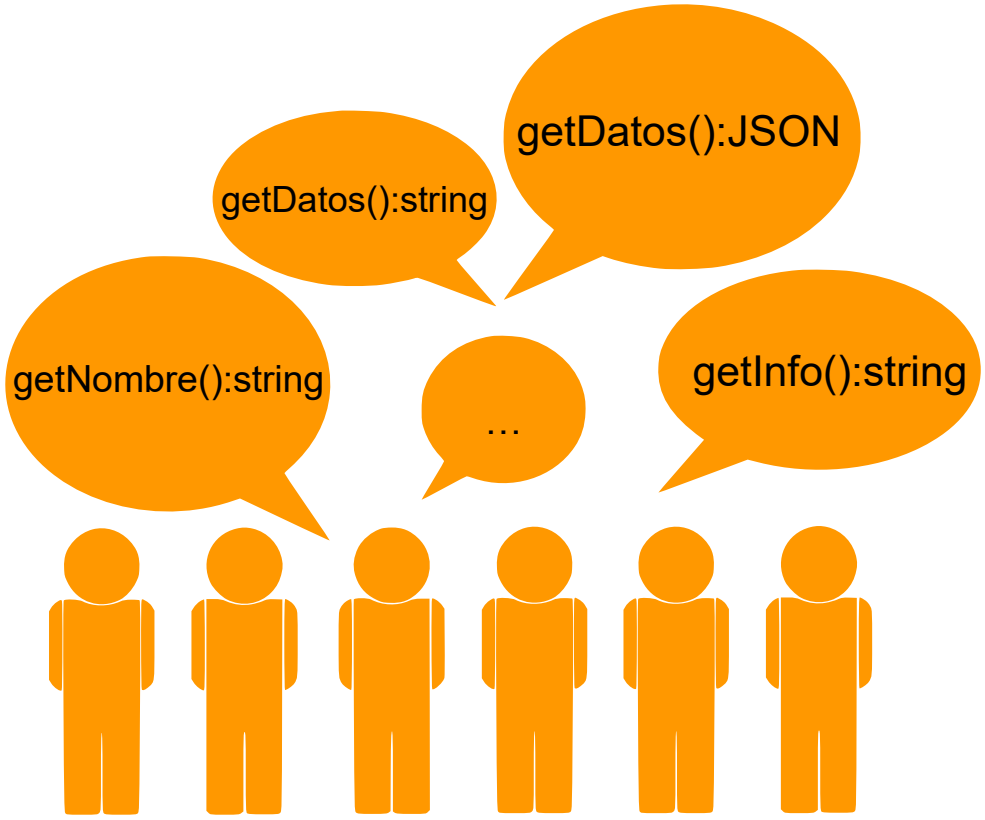


Diagrama de comunicación

- Cada persona realiza el diagrama de comunicación de las funcionalidades que le corresponden
  - Cada persona obtiene el diagrama de clases necesario para implementar sus funcionalidades
- Se genera un único diagrama de clases
  - Se unifican métodos, parámetros, etc.
  - Se toman decisiones respecto a las relaciones entre clases
- Se rehacen los diagramas de comunicación para ajustarlos al diagrama de clases común

Diagrama de comunicación

- Se pueden hacer patentes nuevas necesidades de relación entre las clases

- Se va analizar teniendo en mente la separación entre
  - Vista (Interfaces)
  - Modelo (Datos)
  - Controlador (Lógica)

Trabajaremos con una adaptación de los patrones
MVC (Modelo – Vista – Controlador)
MVP (Modelo – Vista – Presentador)

Diagrama de comunicación

- Vamos a trabajar pensando en un desarrollo en forma de API dividida en frontend y backend
  - Cada llamada debe ser "autosuficiente", no hay memoria de lo ejecutado anteriormente

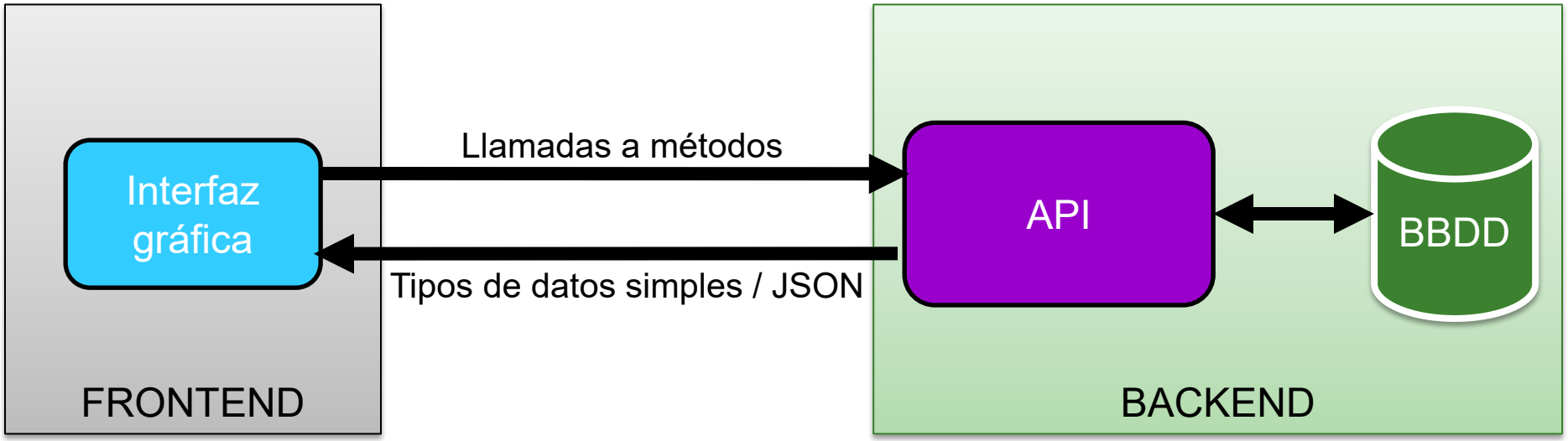


Diagrama de comunicación

- Las clases de la vista sólo deben trabajar con tipos de datos simples
- De este modo, son independientes
  - No hay que modificarlas si cambia el funcionamiento interno
  - Se pueden reutilizar
  - Se pueden desarrollar distintas interfaces intercambiables
    - Una web
    - Una app
    - Un programa Java

Diagrama de comunicación

- Para devolver información estructurada a la Vista usaremos JSON
- JSON es un formato textual formado por pares *nombre : valor* entre llaves
  - El nombre no es obligatorio
- Los valores pueden ser
  - Entero: { edad: 21 }
  - String: { nombre: “Iker” }
  - Boolean: { aprobado: True }
  - Array: { aficiones: [“cine”, “deporte”, “música”] }
  - Objeto JSON: { est: {nombre: “Iker” , edad: 21, aprobado: True} }

LA VISTA (LA INTERFAZ) NO DEBERÍA TRABAJAR CON OBJETOS

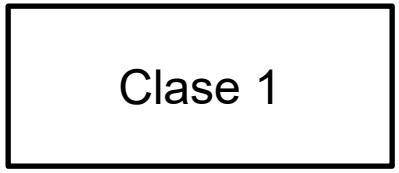
Diagrama de comunicación

- Vamos a trabajar con una separación Modelo-Vista-Controlador estricta
  - Facilitará la modificación de alguna parte sin afectar al resto

→ Interacciones permitidas

Diagrama de comunicación

- En el Diagrama de comunicación se representa
  - Las clases involucradas en la ejecución de un Caso de Uso



  - Las comunicaciones que se producen entre dichas clases

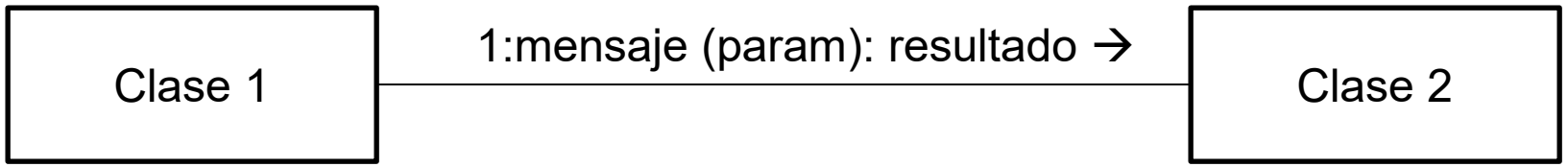


Diagrama de comunicación

- Las comunicaciones siguen el formato:

  orden : nombre (parámetros)

  - Orden: Número que indica el orden de ejecución
  - Nombre: nombre de la operación
  - Parámetros: parámetros necesarios en la operación

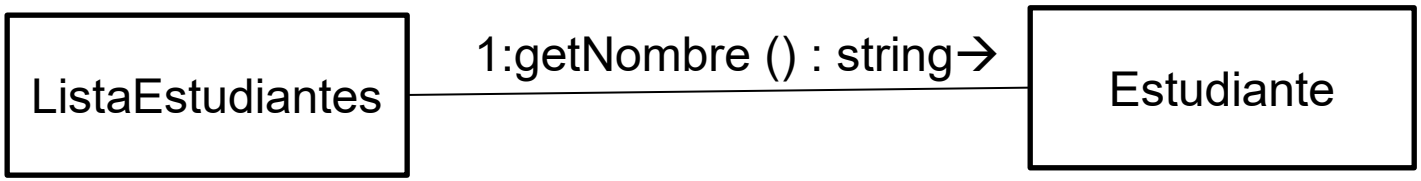
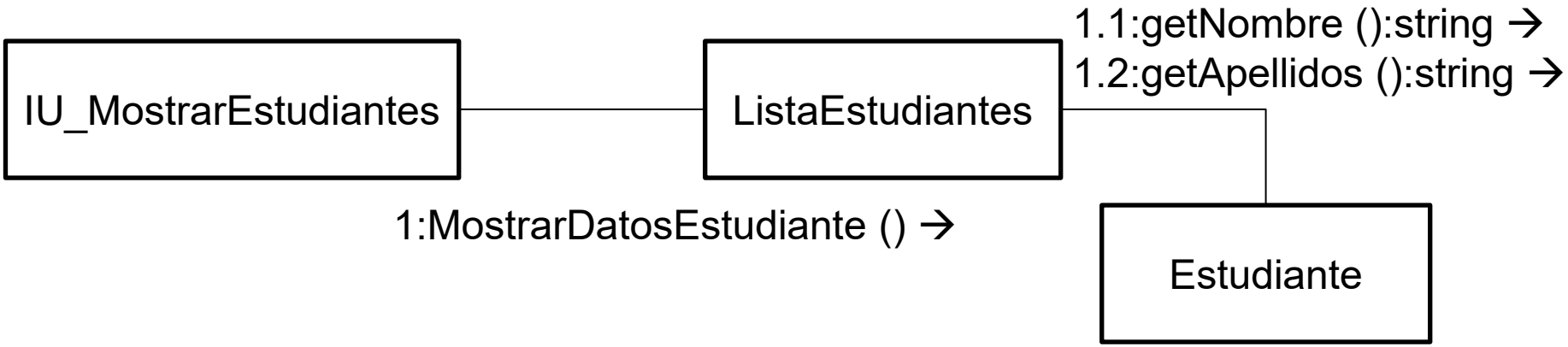


Diagrama de comunicación

- El orden puede ir anidándose para indicar "subpasos" de una operación




```
public void MostrarDatosEstudiante(){
String Nom;
String Ap;
...
   Nom= est.getNombre();
   Ap= est.getApellidos();
...
}
```

```
public String getNombre(){
    return this.nombre;
}

public String getApellidos(){
    return this.apellidos;
}
```

Diagrama de comunicación

- Las interacciones de los actores también aparecen en el diagrama de comunicación

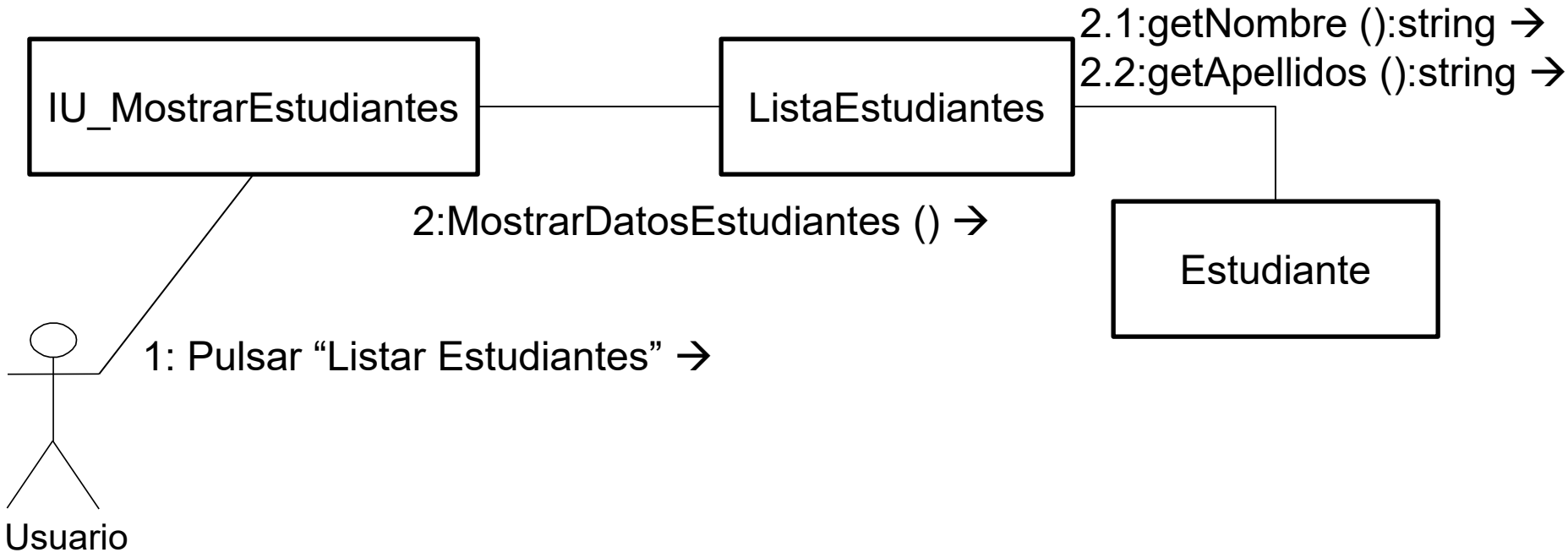
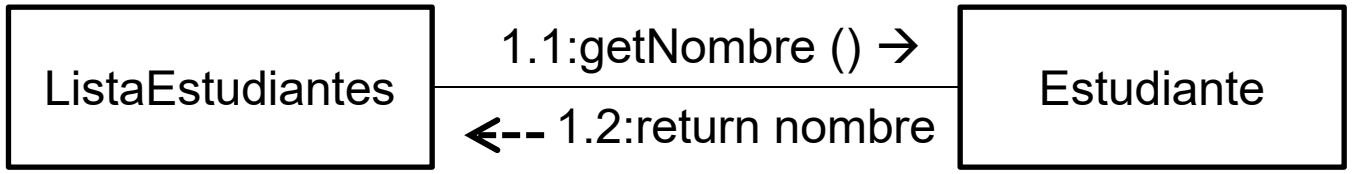


Diagrama de comunicación

- Se puede indicar el resultado de una operación
  - Se recomienda usarlo solo cuando se considera estrictamente necesario para una mejor comprensión del diagrama



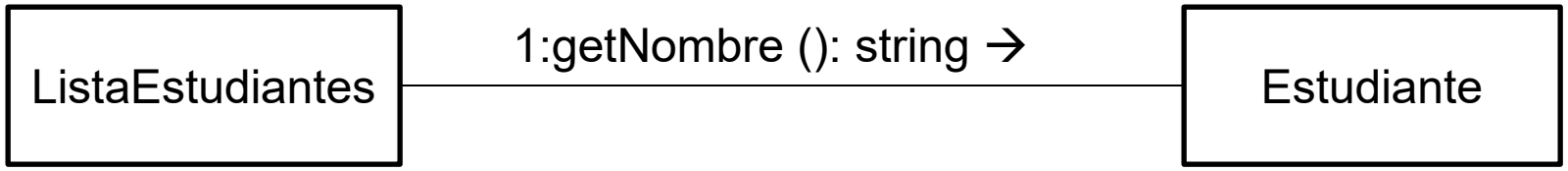


Diagrama de comunicación

- En las devoluciones a la interfaz especificaremos la estructura del JSON

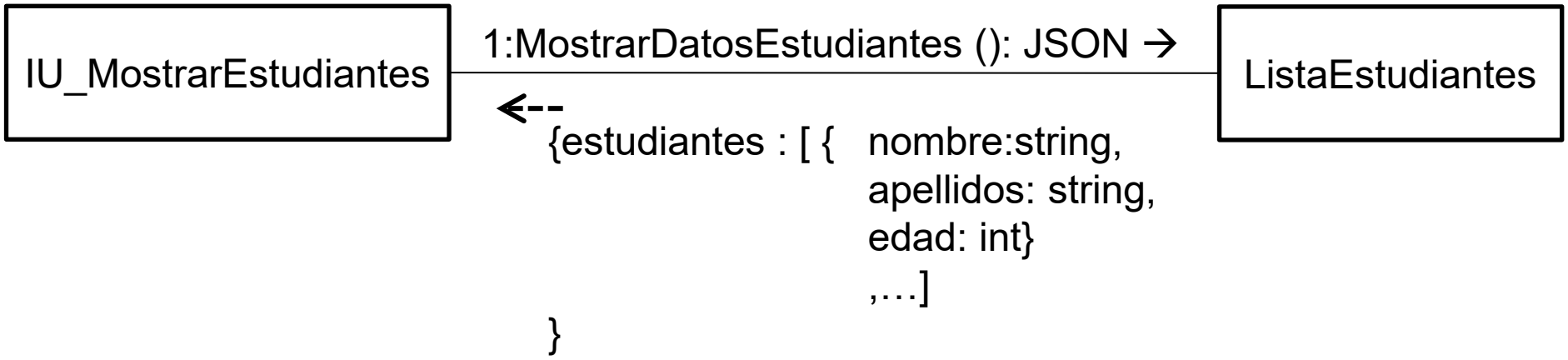


Diagrama de comunicación

- El resultado de una operación se puede recoger en una variable y usarla más adelante

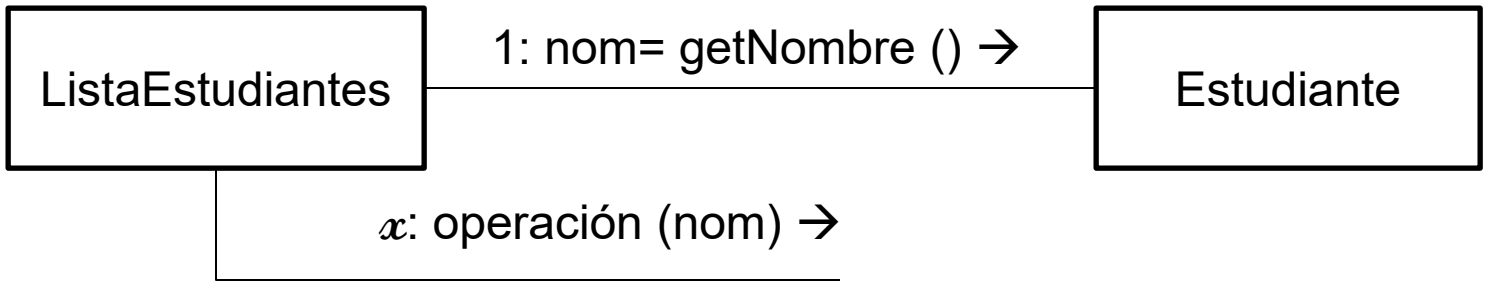


Diagrama de comunicación

- Se pueden indicar condiciones, pero es mejor hacer diagramas de comunicación distintos para cada situación

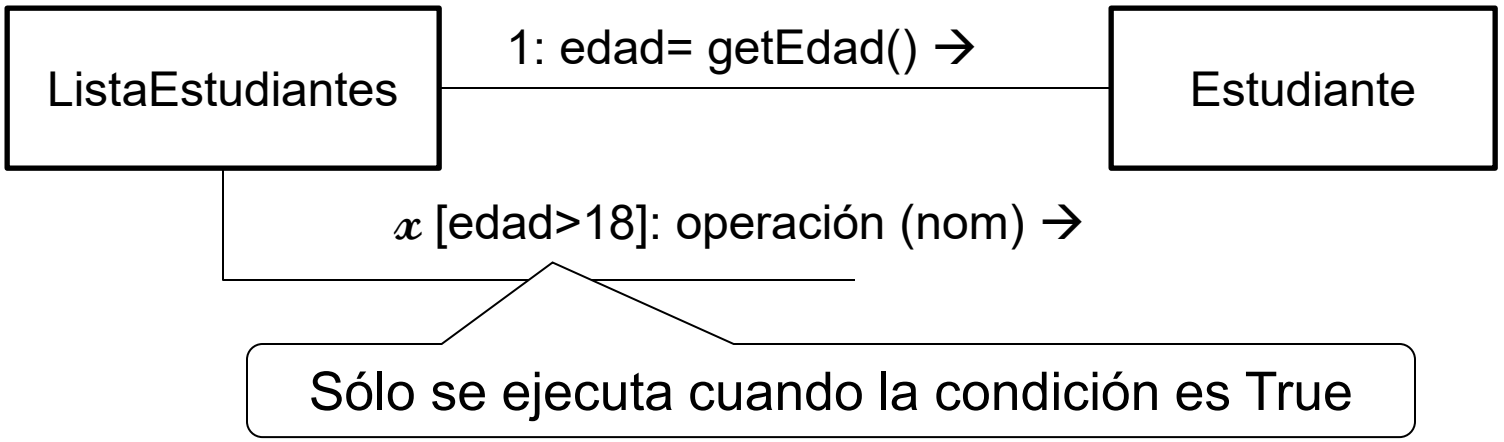


Diagrama de comunicación

- También se pueden indicar repeticiones

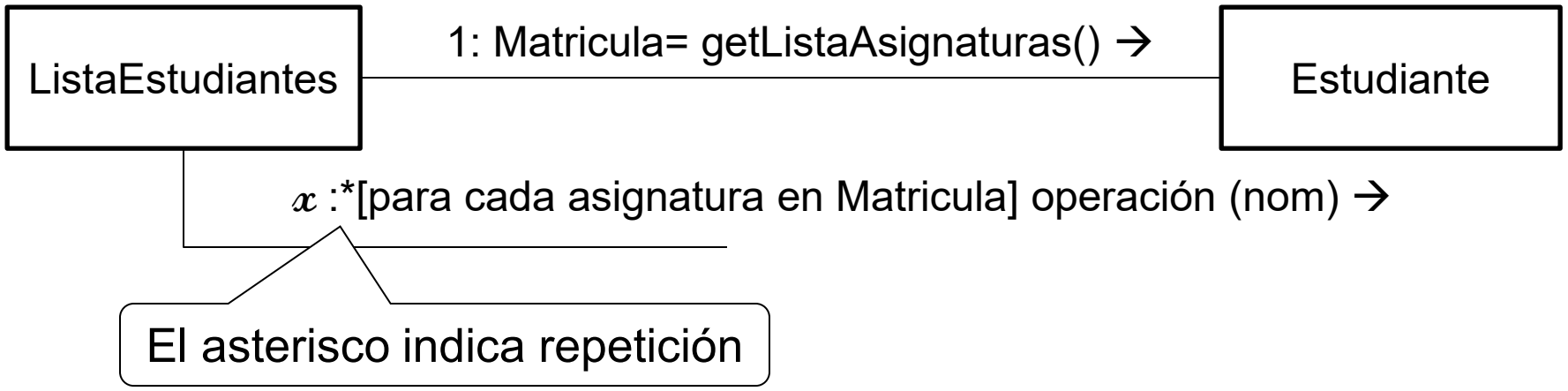


Diagrama de comunicación. Ejemplo

- Caso de uso : Tomar libro prestado
  - Flujo de eventos:
    - El usuario introduce la signatura del libro y su número de socio
    - Si no hay ninguna copia libre
      - Se le avisa al usuario
    - Si hay alguna copia libre
      - Si el usuario ha llegado al máximo de prestamos
        - Se le avisa
      - Si no ha llegado al máximo
        - Se almacena el nuevo préstamo y se le muestra por pantalla el código de la copia que se le presta

Diagrama de comunicación. Ejemplo

- Caso de uso : Tomar libro prestado
  - Prototipo de la interfaz

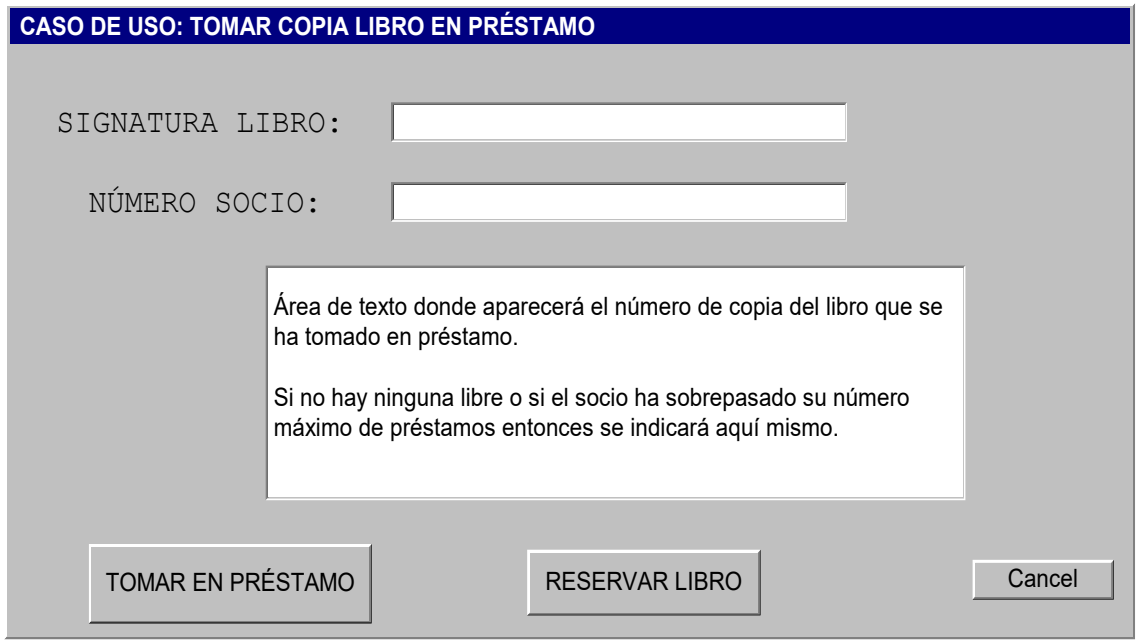


Diagrama de comunicación. Ejemplo

- Caso de uso : Tomar libro prestado
  - Modelo del dominio

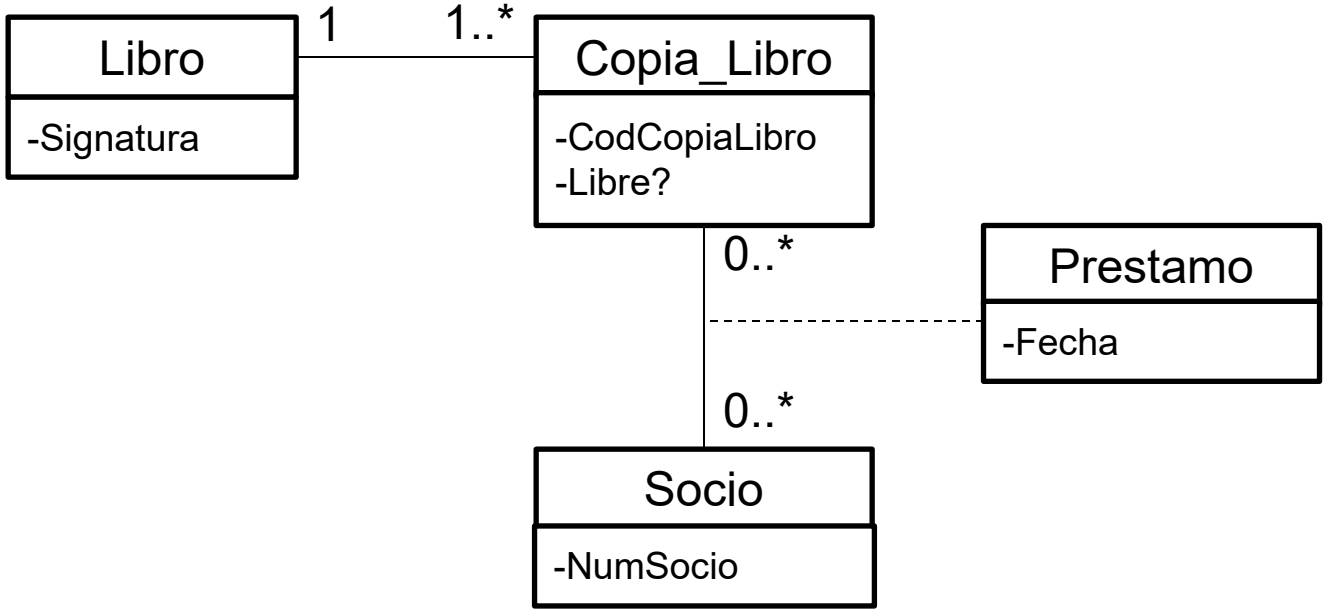
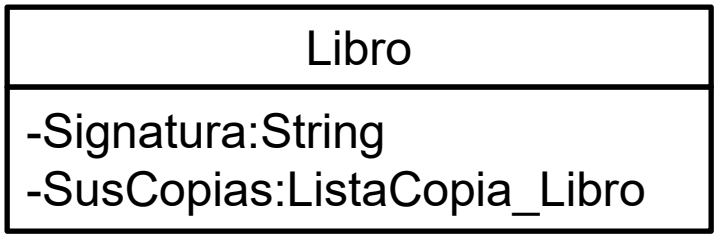
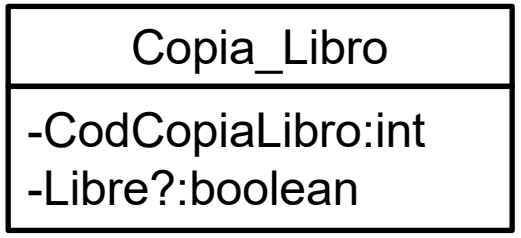


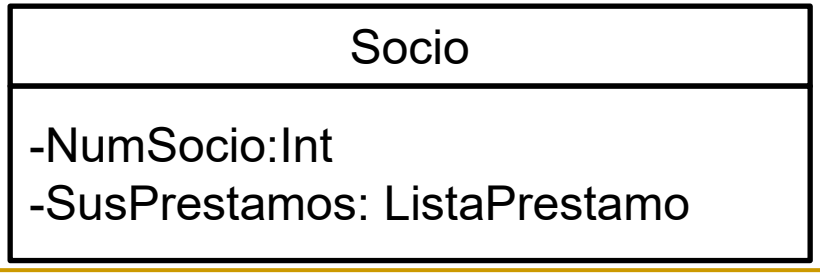
Diagrama de comunicación. Ejemplo

- Caso de uso : Tomar libro prestado
  - Voy a ignorar la persistencia de la información y voy a suponer que tengo toda la información cargada en objetos
    - Diagrama de clases

El Modelo







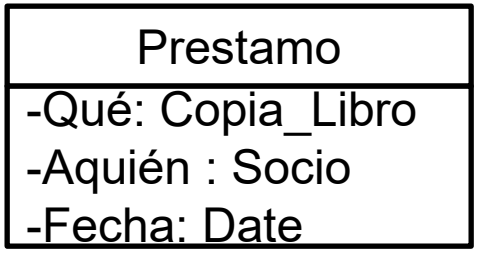
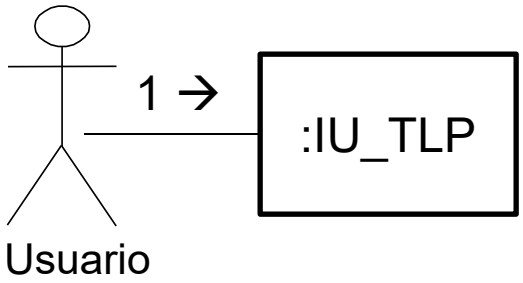


Diagrama de comunicación. Ejemplo

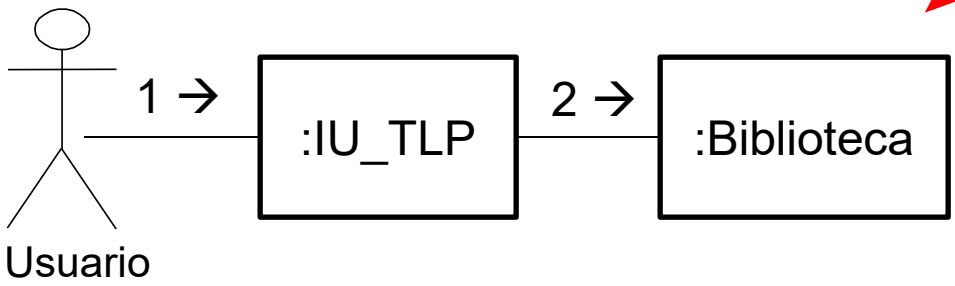
- Caso de uso : Tomar libro prestado
  - Falta el Controlador
    - Serán las clases que contengan la lógica del caso de uso
    - Recogerán lo que ocurra en la interfaz y trabajarán con el modelo
    - Tendrán una única instancia (MAEs)
    - Pueden contener una lista de objetos que gestionar
    - Cuántas y cuales poner depende del diseño

Diagrama de comunicación. Ejemplo



1: Introduce datos y pulsa “Tomar en Préstamo”

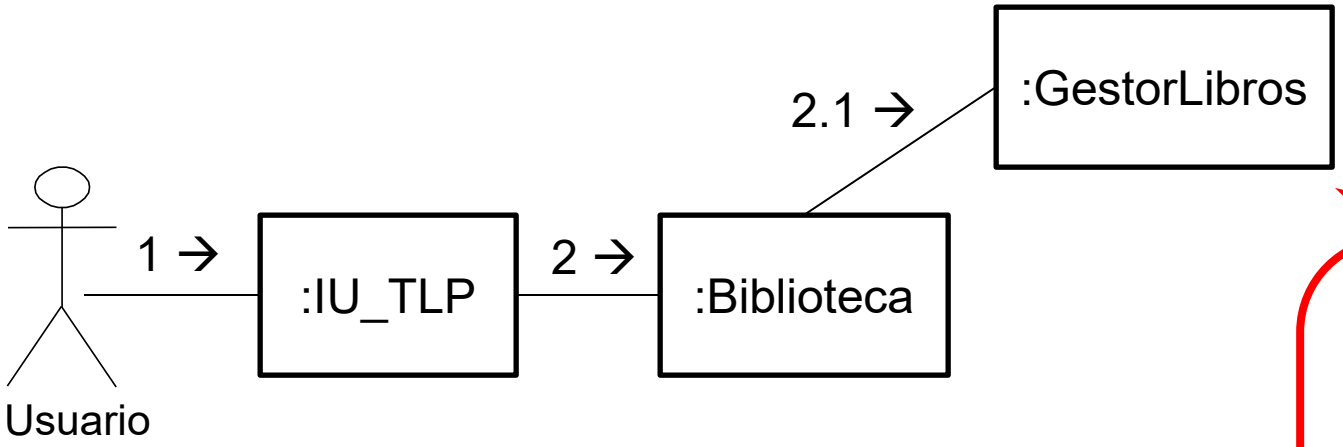
Diagrama de comunicación. Ejemplo



Usamos una única clase que actuará de frontera entre la Vista y el Controlador. Será la única que conozca “las tripas” del controlador y facilitará el mantenimiento y la ampliación del sistema

1: Introduce datos y pulsa “Tomar en Préstamo”
2: TomarLibroPrestamo (signatura,numsocio)

Diagrama de comunicación. Ejemplo

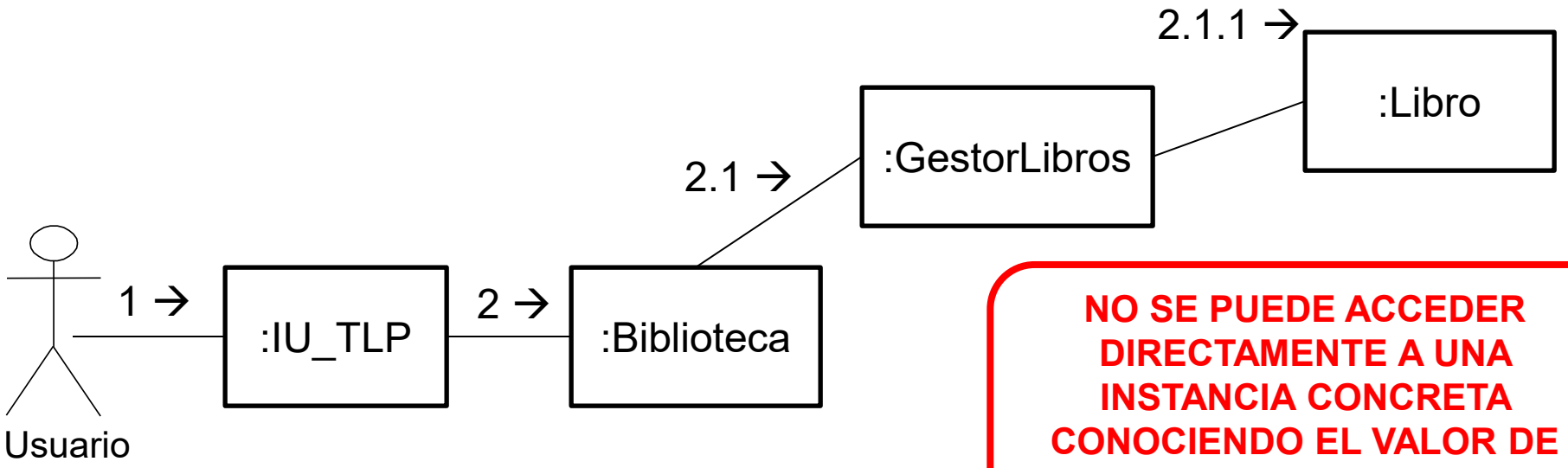


1: Introduce datos y pulsa “Tomar en Préstamo”
2: TomarLibroPrestamo (signatura,numsocio)
2.1: laCopiaLibre=BuscarCopiaLibre (signatura)

Para encontrar una copia a partir de la signatura, primero hay que encontrar el libro.

El GestorLibros conoce la referencia a todos los libros que existen en el sistema (la clase necesita un atributo para agruparlos)

Diagrama de comunicación. Ejemplo

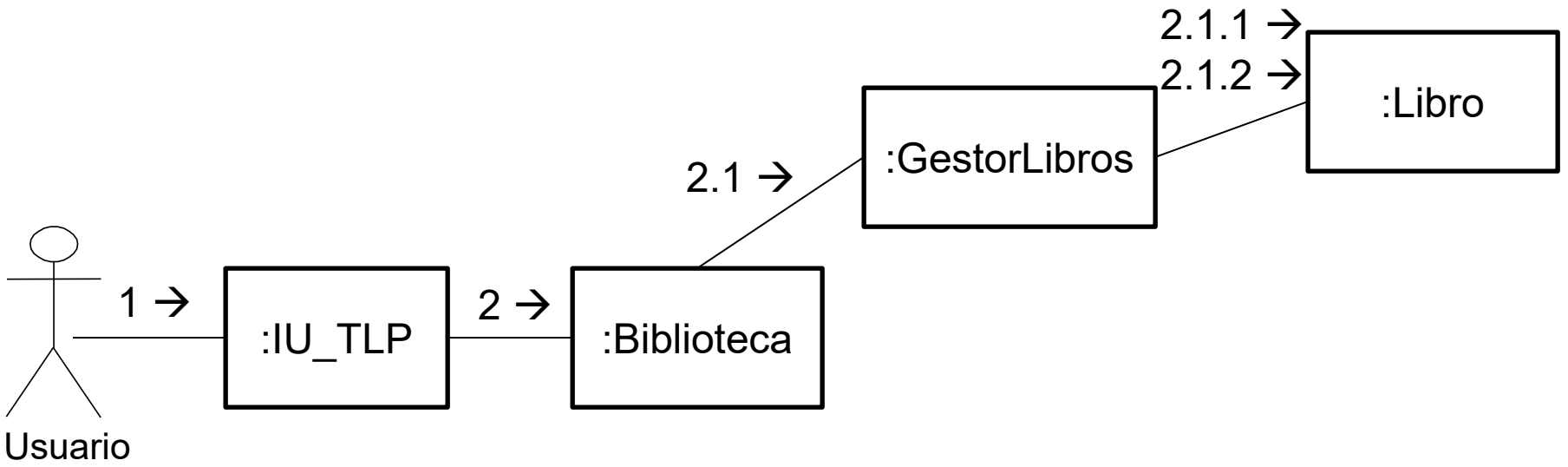


NO SE PUEDE ACCEDER DIRECTAMENTE A UNA INSTANCIA CONCRETA CONOCIENDO EL VALOR DE UNO DE SUS ATRIBUTOS

1: Introduce datos y pulsa “Tomar en Préstamo”
2: TomarLibroPrestamo (signatura,numsocio)
    2.1: laCopiaLibre=BuscarCopiaLibre (signatura)
        2.1.1: getSignatura() : String

Se recorren todas las instancias que existan de Libro obteniendo su signatura y comparándola con la dada para encontrar la instancia concreta con la que tenemos que trabajar

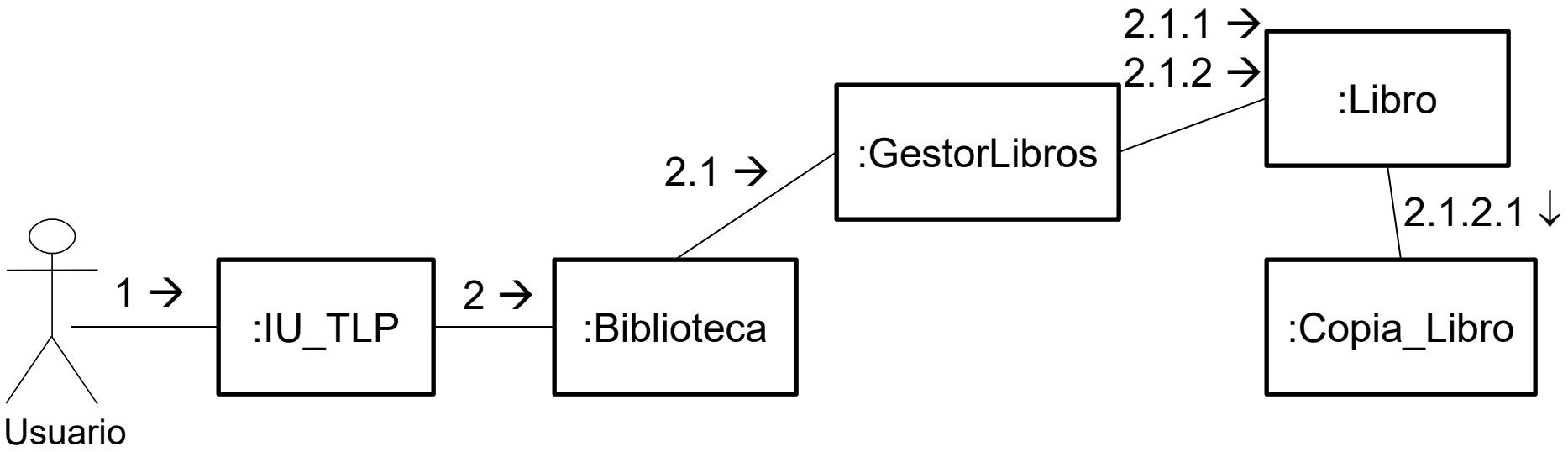
Diagrama de comunicación. Ejemplo



1: Introduce datos y pulsa “Tomar en Préstamo”
2: TomarLibroPrestamo (signatura,numsocio)
    2.1: laCopiaLibre=BuscarCopiaLibre (signatura)
        2.1.1: getSignatura() : String
        2.1.2: ObtenerCopiaLibre(): Copia_Libro

Una vez encontrada la instancia concreta, ejecutamos una operación que nos devuelva una copia de ese libro que esté libre. Podemos hacerlo así porque el Libro tiene la referencia a SusCopias

Diagrama de comunicación. Ejemplo



1: Introduce datos y pulsa “Tomar en Préstamo”
2: TomarLibroPrestamo (signatura,numsocio)
    2.1: laCopiaLibre=BuscarCopiaLibre (signatura)
        2.1.1: getSignatura() : String
        2.1.2: ObtenerCopiaLibre(): Copia_Libro
            2.1.2.1: getLibre?() : boolean

La operación de la clase Libro recorrerá las instancias recogidas en el atributo SusCopias hasta que encuentre una libre

Diagrama de comunicación. Ejemplo

1: Introduce datos y pulsa “Tomar en Préstamo”
2: TomarLibroPrestamo (signatura,numsocio)
    2.1: laCopiaLibre=BuscarCopiaLibre (signatura)
        2.1.1: getSignatura() : String
        2.1.2: ObtenerCopiaLibre(): Copia_Libro
            2.1.2.1: getLibre?() : boolean
    2.2: elSocio= BuscarSocio (numsocio)
        2.2.1: getNumSocio() :int
    2.3: ComprobarNumeroPrestamos (elSocio)
        2.3.1: getMaximo?() :boolean
    2.4: AlmacenarPrestamo (elSocio,laCopiaLibre)
        2.4.1: new (elSocio, laCopiaLibre, now())
        2.4.2: actualizarListaPrestamos (laCopiaLibre)
        2.4.3: setLibre(False)
    2.5: ObtenerCodCopia (laCopiaLibre)
        2.5.1: getCodCopia(): int

Hay que buscar el objeto que se corresponde a ese numsocio

Comprobando cuántos prestamos tiene en su lista

La fecha actual siempre está disponible

Para poder mostrarlo en la interfaz

Hay que actualizar la lista de préstamos de el socio

Diagrama de comunicación. Ejemplo

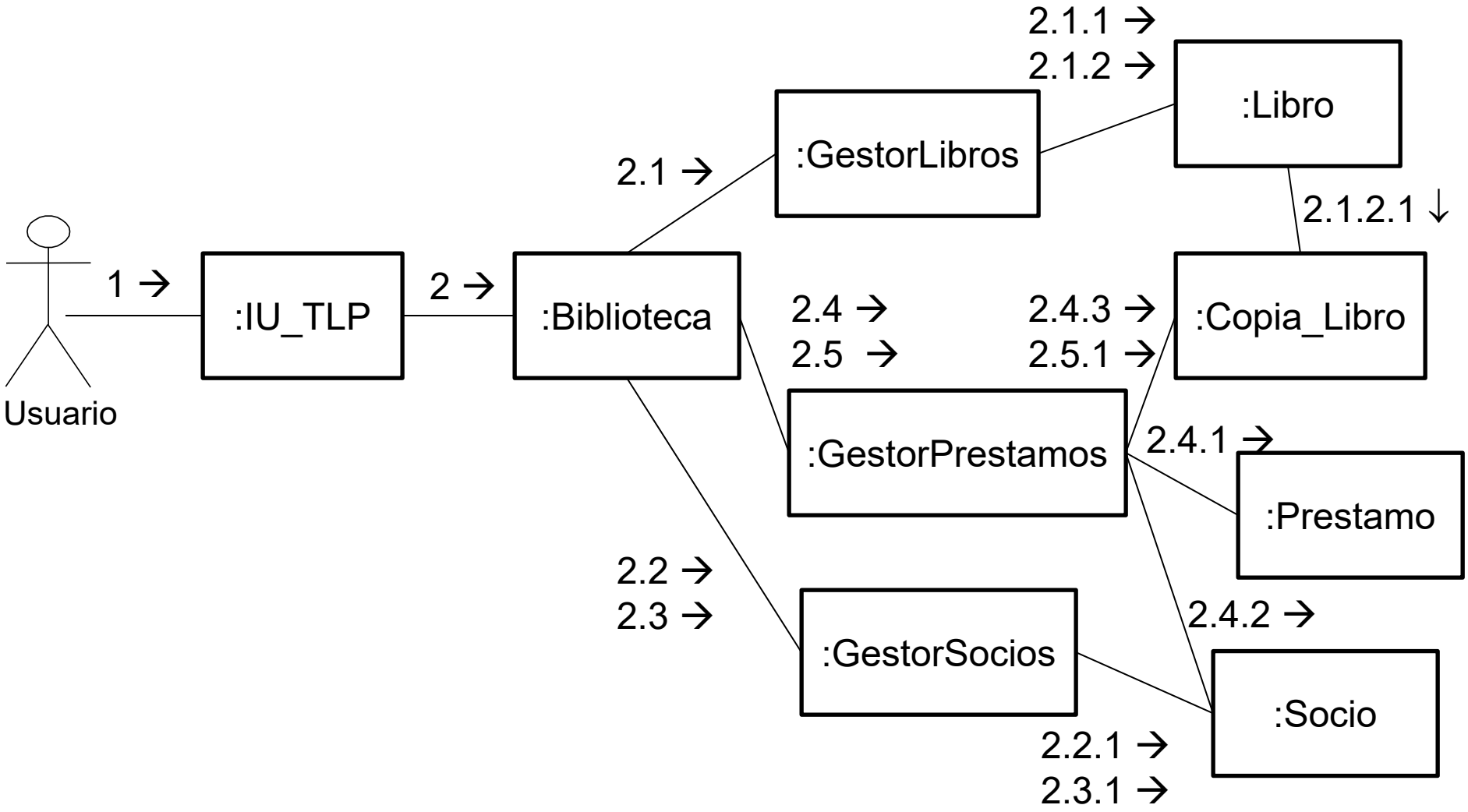


Diagrama de comunicación. Ejemplo

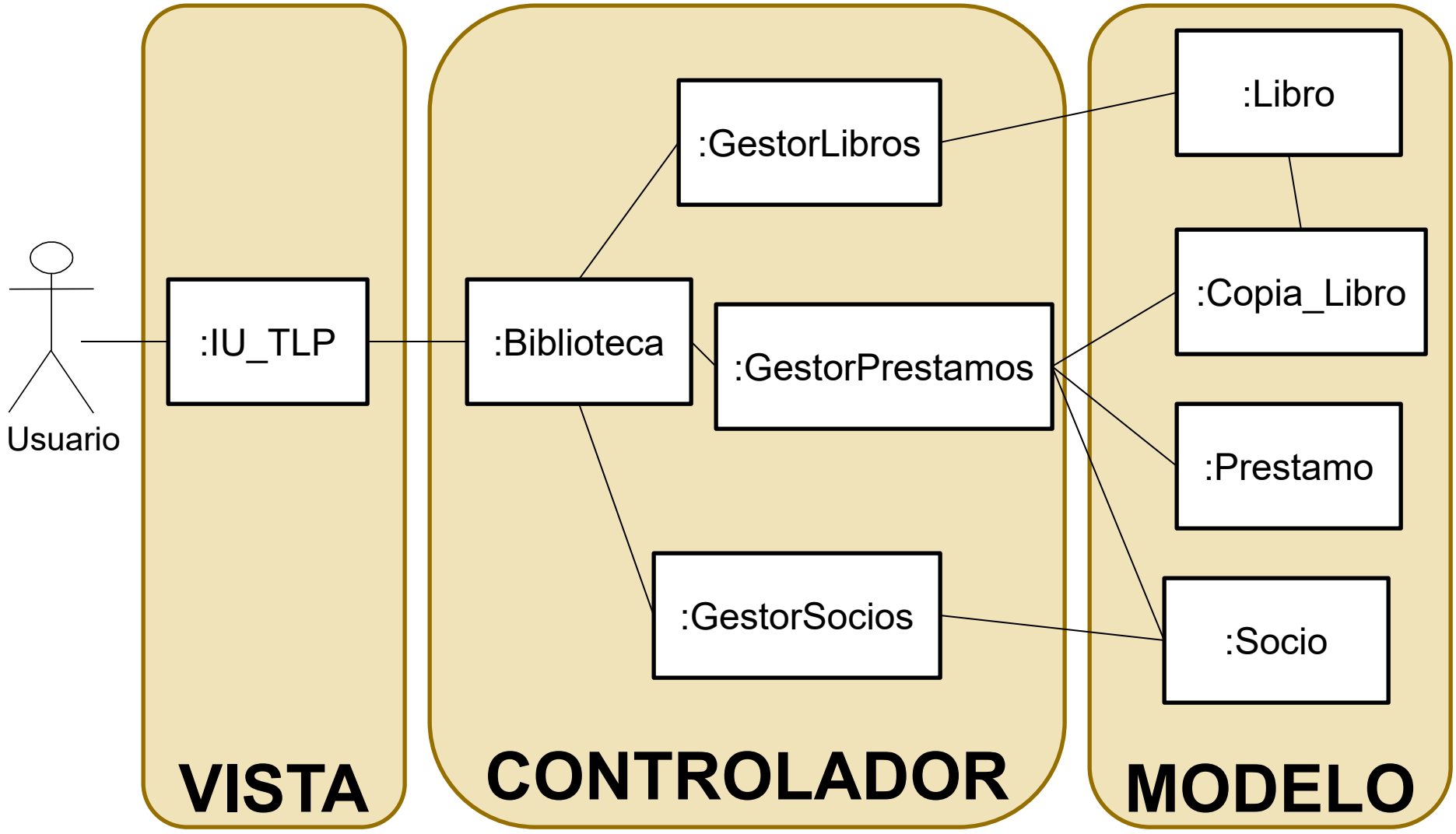
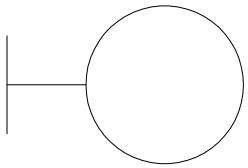
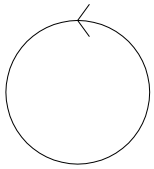


Diagrama de comunicación

- Se pueden usar estereotipos de las clases, y de este modo indicar qué tipo de clases son
- Existen 3 estereotipos:
  - Clase Frontera (Vista)



  - Clase Control (Controlador)



  - Clase Entidad (Modelo)

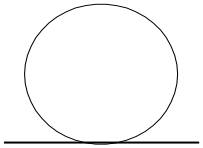


Diagrama de comunicación. Ejemplo

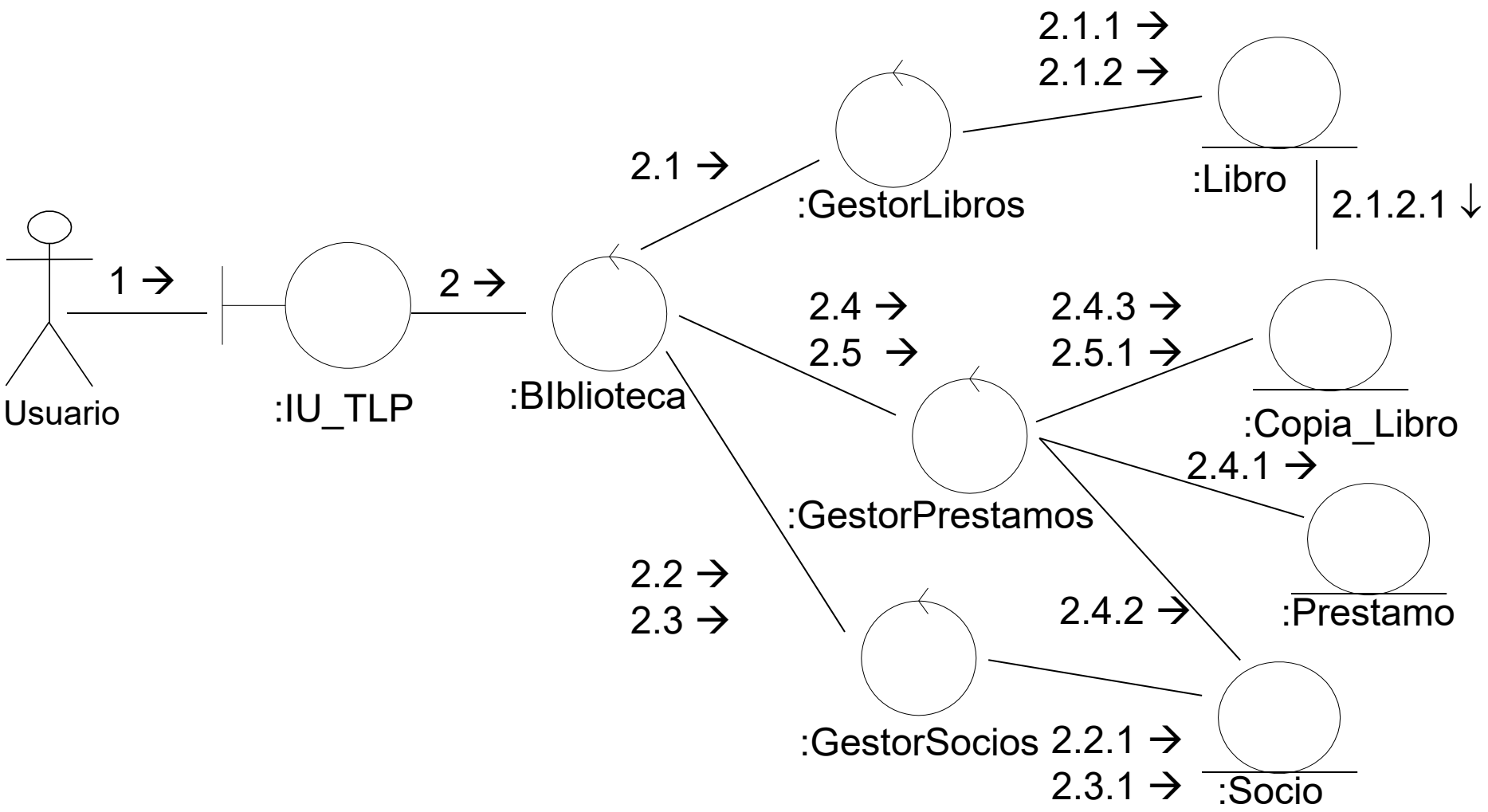


Diagrama de comunicación

- A partir del diagrama de comunicación obtenemos el diagrama de clases necesario para ese caso de uso

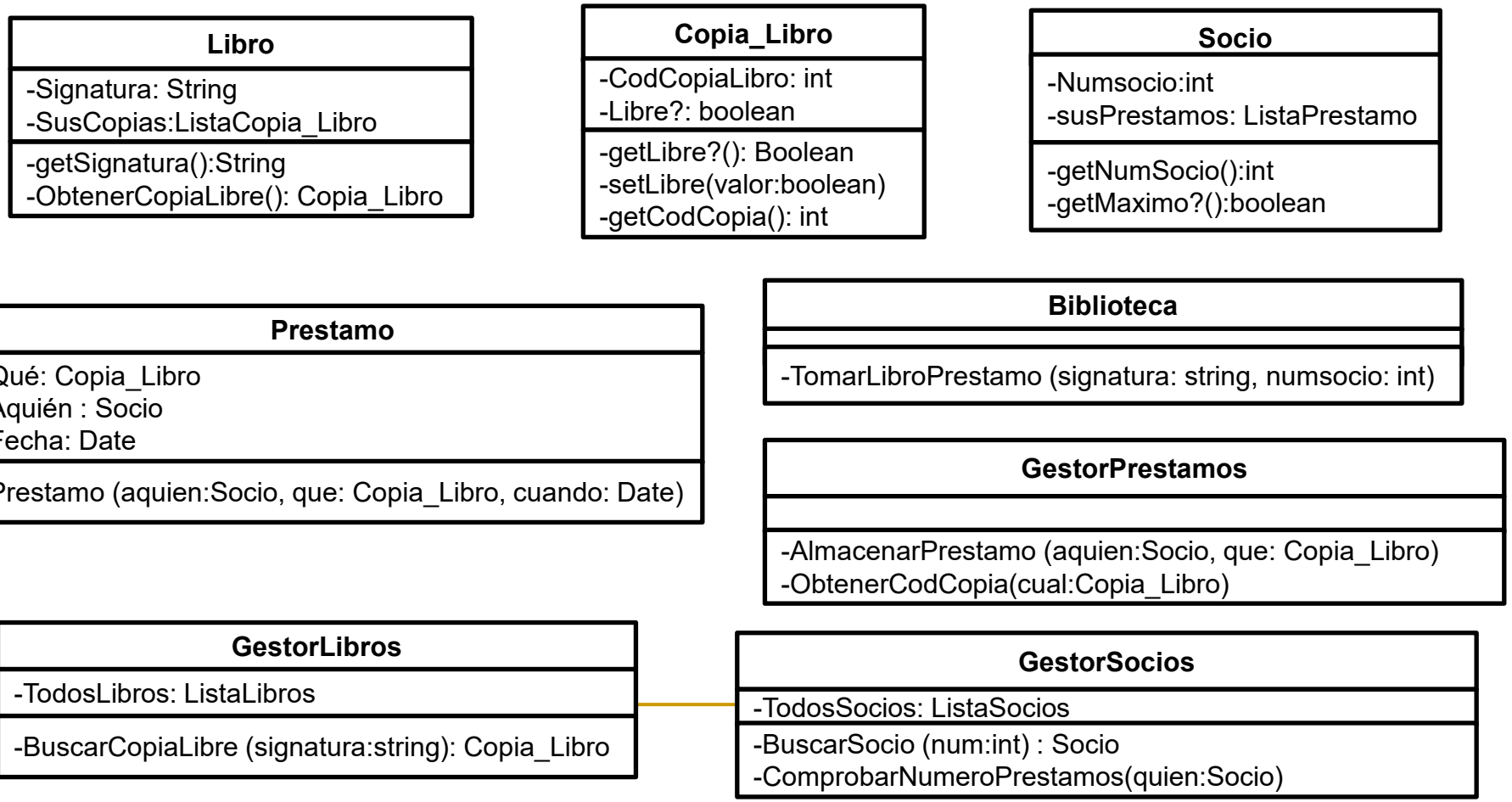


Diagrama de comunicación

- A partir del diagrama de clases necesario para cada caso de uso
  - Se unifican tomando decisiones
    - Dónde colocar los atributos
    - Qué clases de control utilizar
    - Qué operaciones situar en cada clase de control
    - Qué operaciones se pueden reutilizar
- Se obtiene un único diagrama de clases para todo el sistema

Modelo usando un SGBD relacional

- La Orientación a Objetos no proporciona persistencia de los datos
- Para lograr persistencia incluiremos una BBDD
- Nuestros diagramas serán independientes del SGBD relacional que usemos para almacenar los datos
  - MySQL
  - Oracle
  - Access
  - ….

Modelo usando un SGBD relacional

- Cuando el sistema arranca, los datos están almacenados en la Base de Datos
- Cuando el sistema finaliza, los datos tienen que estar actualizados en la Base de Datos
- Cuando el sistema está funcionando, los datos pueden estar
  - Cargados en objetos
  - Almacenados en la Base de Datos

Modelo usando un SGBD relacional

- Elegir el momento en el que los datos de la Base de Datos se cargan en los objetos (si se hace) es una decisión del análisis/diseño
- Elegir cuándo se actualizan los datos de los objetos en la Base de Datos (es obligatorio hacerlo) es decisión del análisis/diseño
- Razones a tener en cuenta
  - Eficiencia
  - Necesidades de actualización en tiempo real

Modelo usando un SGBD relacional

- Posibles cargas de datos en objetos desde la BBDD
  - Cargar todos los datos al iniciar el sistema y crear las instancias correspondientes
  - No cargar ningún dato
  - Cargar sólo parte de los datos
    - Los pertenecientes al usuario identificado
    - Los que solicite el usuario
    - ….

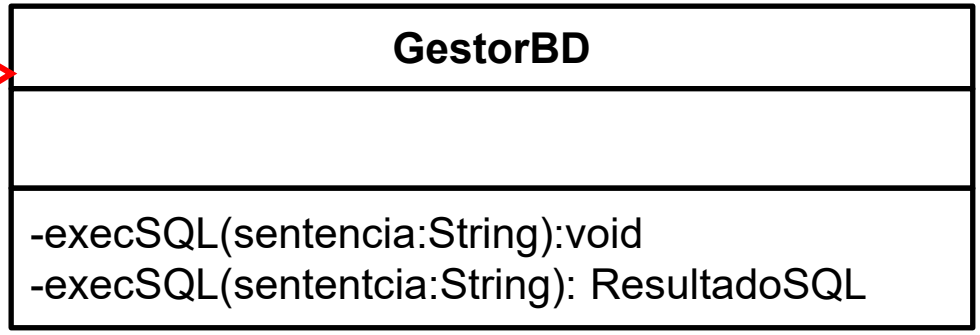
Modelo usando un SGBD relacional

- Posibles formas de trabajo con los datos
  - Si no están cargados en objetos
    - Directamente contra la Base de Datos
  - Si están cargados en objetos
    - Trabajar sólo con los objetos
    - Trabajar con los objetos y actualizar en ese momento la Base de Datos (creando, modificando o eliminando información)
- Al cerrar el sistema
  - Si hay datos que no se han actualizado en la Base de Datos, hacerlo

Modelo usando un SGBD relacional

- Para seguir este patrón utilizaremos la clase GestorBD que tiene 2 métodos:
  - Para realizar INSERT/DELETE/UPDATE (no devuelven nada)
  - Para realizar consultas tipo SELECT (deben devolver el resultado de la consulta)

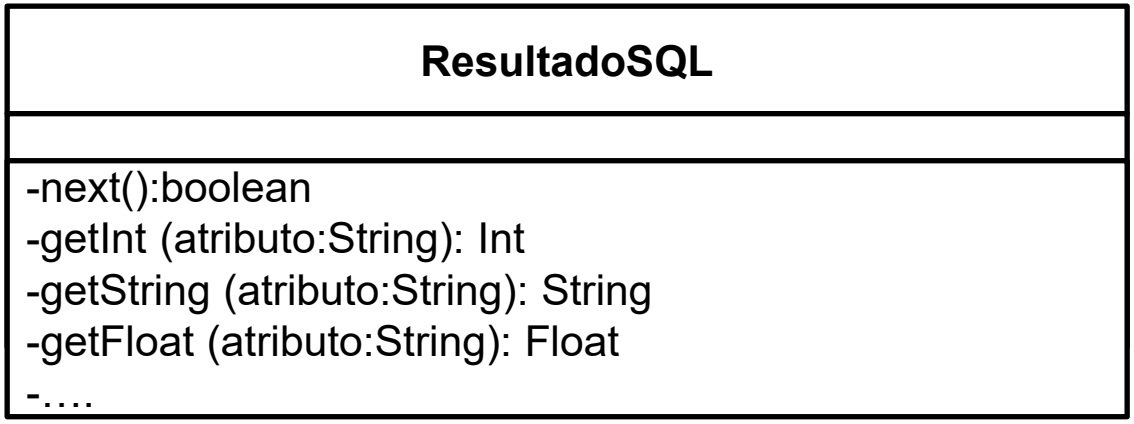
Representa al SGBD. Solo tiene una instancia.



Modelo usando un SGBD relacional

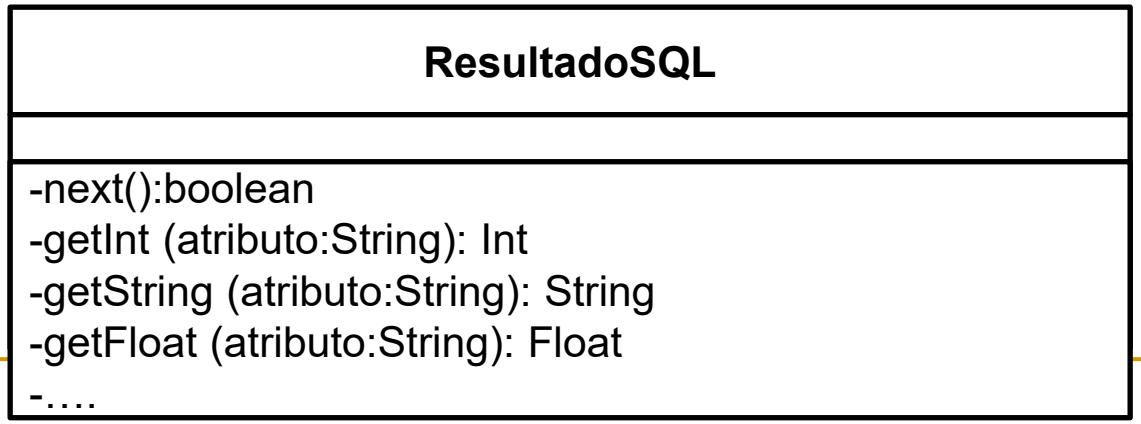
- Para trabajar con el resultado de una pregunta SQL usaremos la clase ResultadoSQL con los métodos (además de la constructora)
  - next() → "selecciona" la siguiente (o primera) tupla del resultado. Devuelve false si no hay más tuplas.

Cada vez que se ejecute una sentencia SQL de tipo SELECT, se genera una instancia



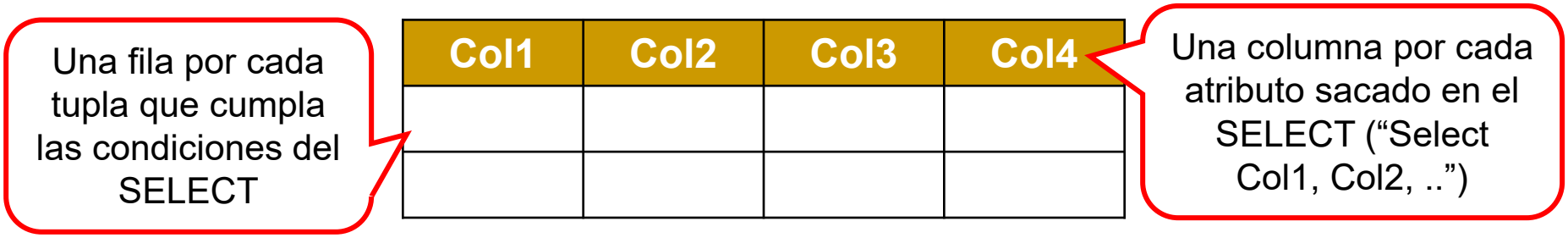
Modelo usando un SGBD relacional

- Para trabajar con el resultado de una pregunta SQL usaremos la clase ResultadoSQL
  - get*TipoDatos* (nombreatributo) → Obtiene el valor de ese atributo en la tupla seleccionada (usando next). Daremos por hecho que hay un método para cada tipo de datos que se puede usar en la BBDD



Modelo usando un SGBD relacional

- Un objeto ResultadoSQL tiene forma de tabla



- Al hacer un getTipoDatos (nombreatributo), nos devolverá el valor de esa columna de la fila apuntada por el next
  - El next comienza apuntando al vacío

Modelo usando un SGBD relacional

- Ejemplo de ResultadoSQL al crearlo

NEXT

Nombre	DNI	Edad
Pepe	45235685Z	25

  - getString("Nombre") → ERROR
  - Next() → True

NEXT

Nombre	DNI	Edad
Pepe	45235685Z	25

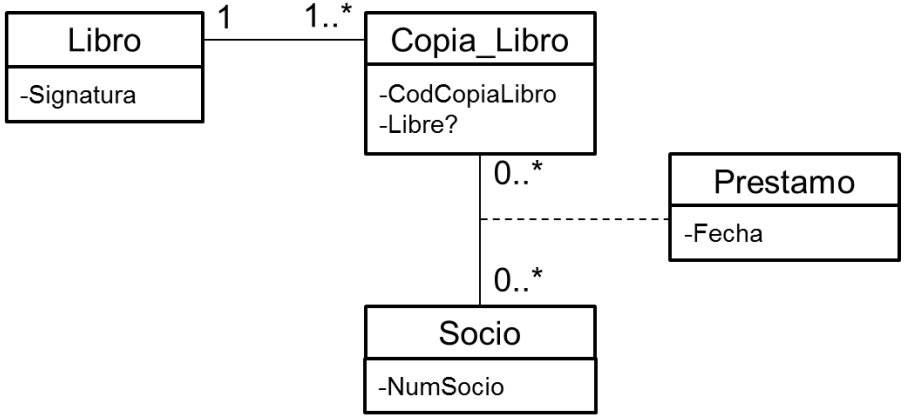
  - getString("Nombre") → Pepe
  - Next() → False

Nombre	DNI	Edad
Pepe	45235685Z	25

NEXT

Diagrama de comunicación. Ejemplo

- Caso de uso : Tomar libro prestado
  - Hay que transformar el modelo del dominio en un esquema relacional



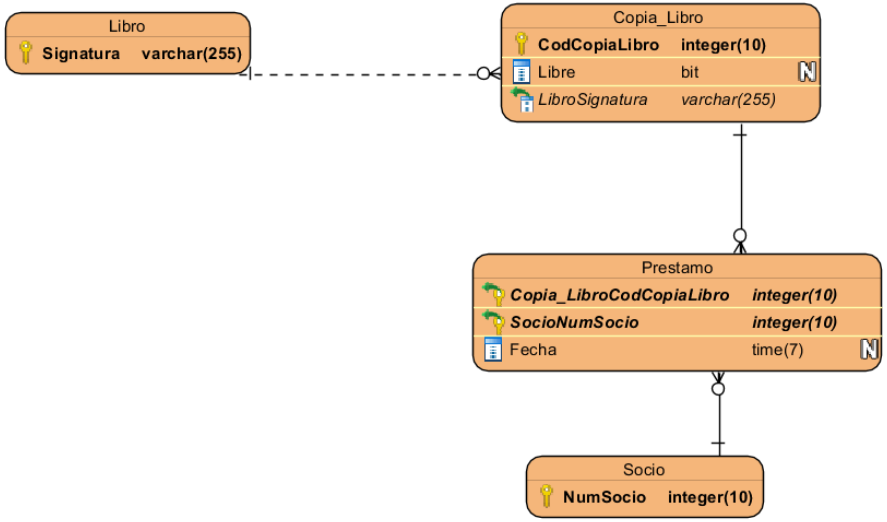
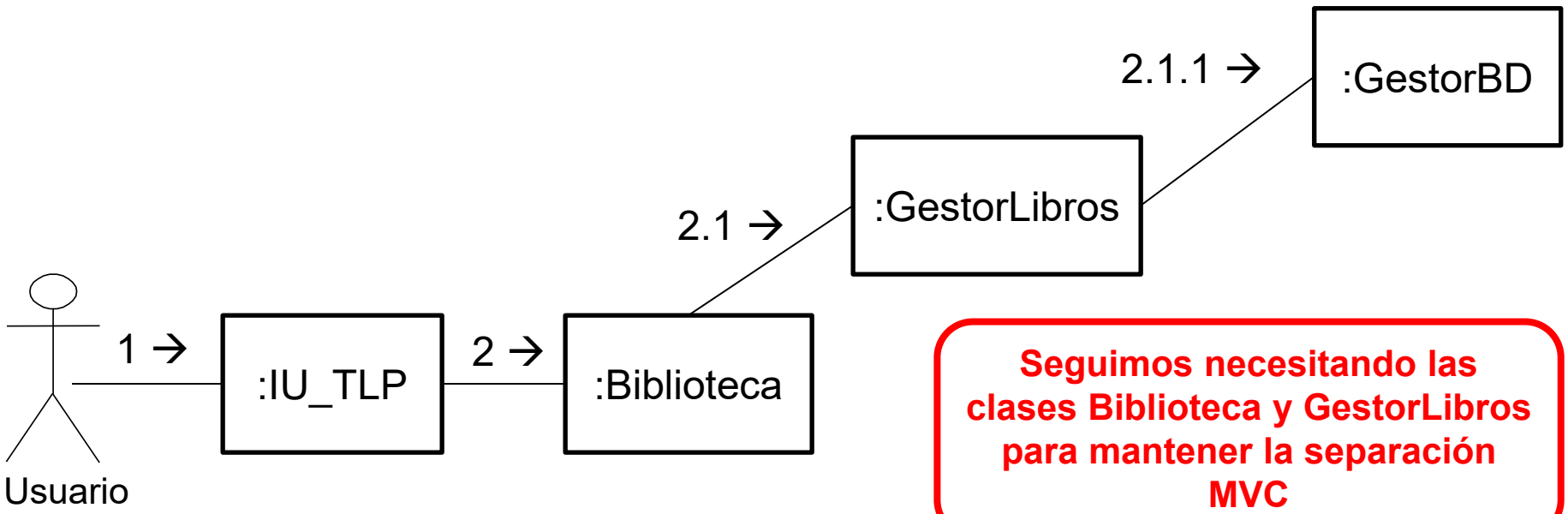


Diagrama de comunicación. Ejemplo

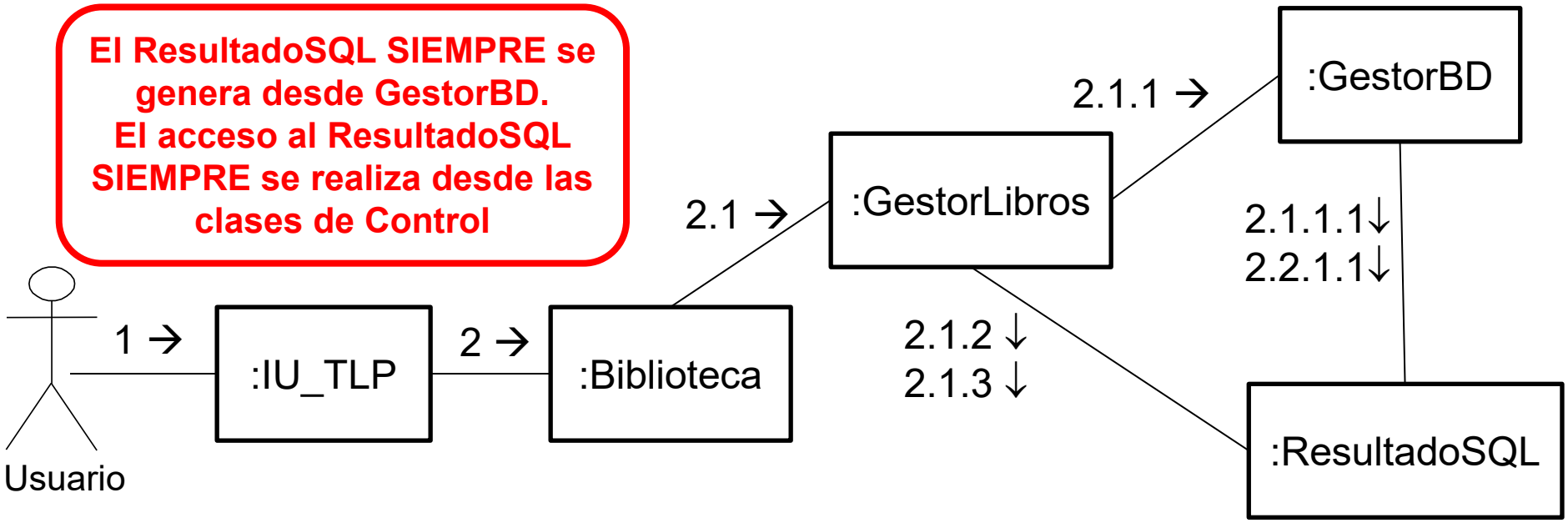


Seguimos necesitando las clases Biblioteca y GestorLibros para mantener la separación MVC

1: Introduce datos y pulsa "Tomar en Préstamo"
2: TomarLibroPrestamo (signatura,numsocio)
    2.1: laCopiaLibre=BuscarCopiaLibre (signatura)
        2.1.1: execSQL("*SELECT CodCopiaLibro FROM Copia_Libro WHERE LibroSignatura= %signatura% AND Libre=1*") : ResultadoSQL

Usando el símbolo % indicamos las variables

Diagrama de comunicación. Ejemplo



1: Introduce datos y pulsa "Tomar en Préstamo"
2: TomarLibroPrestamo (signatura,numsocio)
    2.1: elCodigoCopiaLibre=BuscarCopiaLibre (signatura)
        2.1.1: execSQL("*SELECT CodCopiaLibro FROM Copia_Libro WHERE LibroSignatura= %signatura% AND Libre=1*") : ResultadoSQL
            2.1.1.1: new ResultadoSQL()
        2.1.2: next()
        2.1.3: getInt ("CodCopiaLibro"): int

SIEMPRE hay que hacer un next, aunque sea para colocarnos en la primera tupla

Diagrama de comunicación. Ejemplo

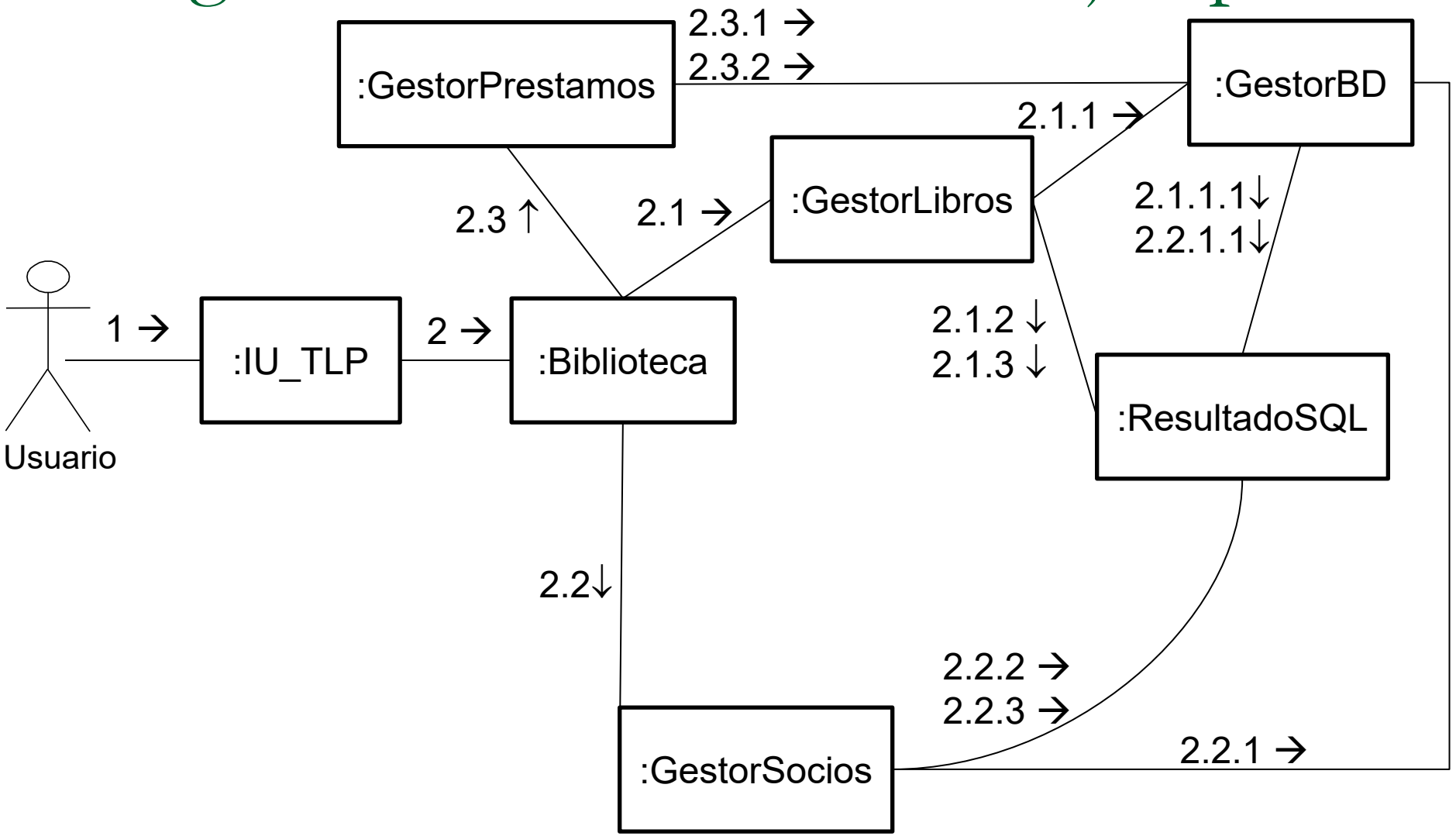


Diagrama de comunicación. Ejemplo

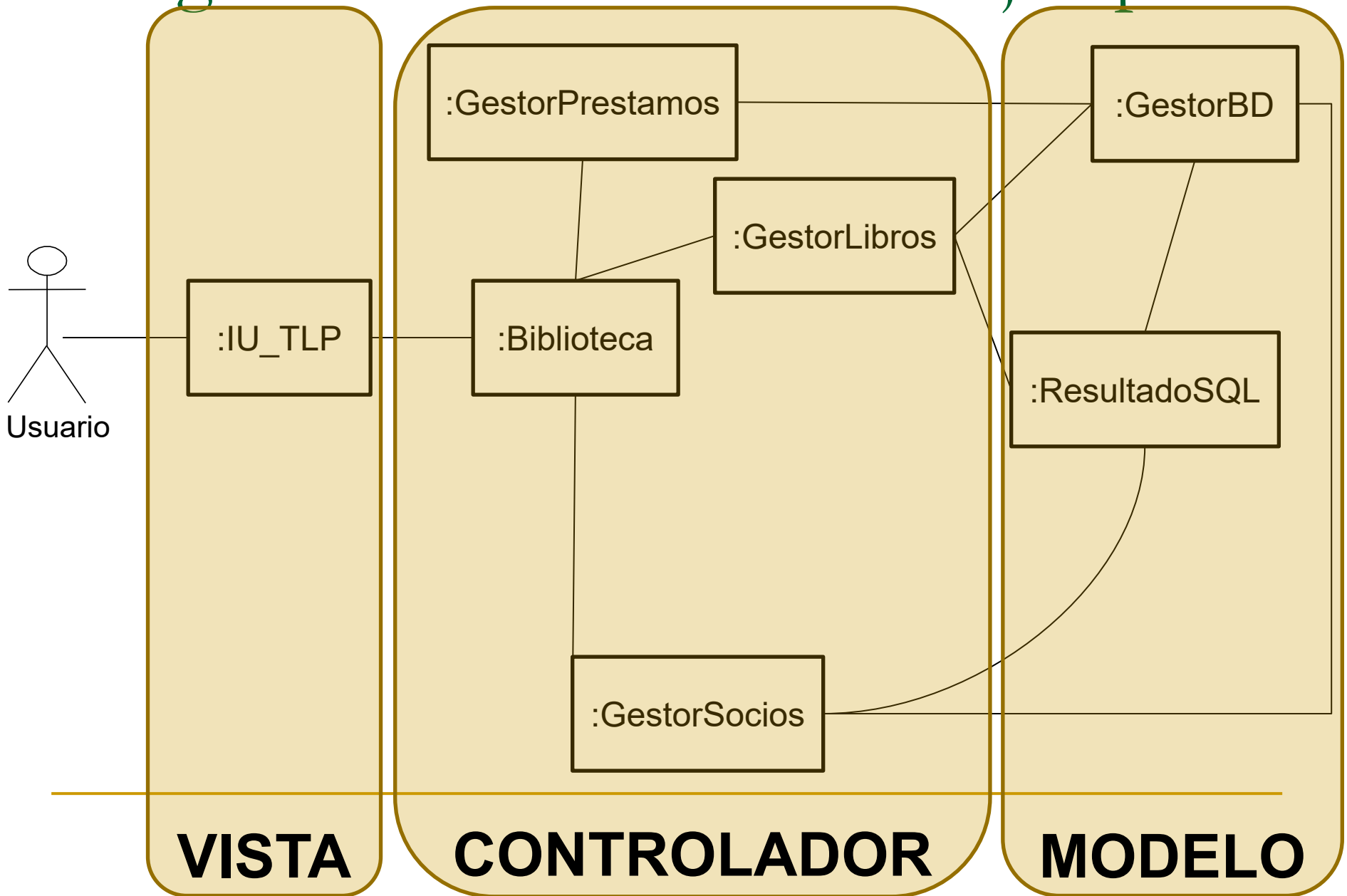


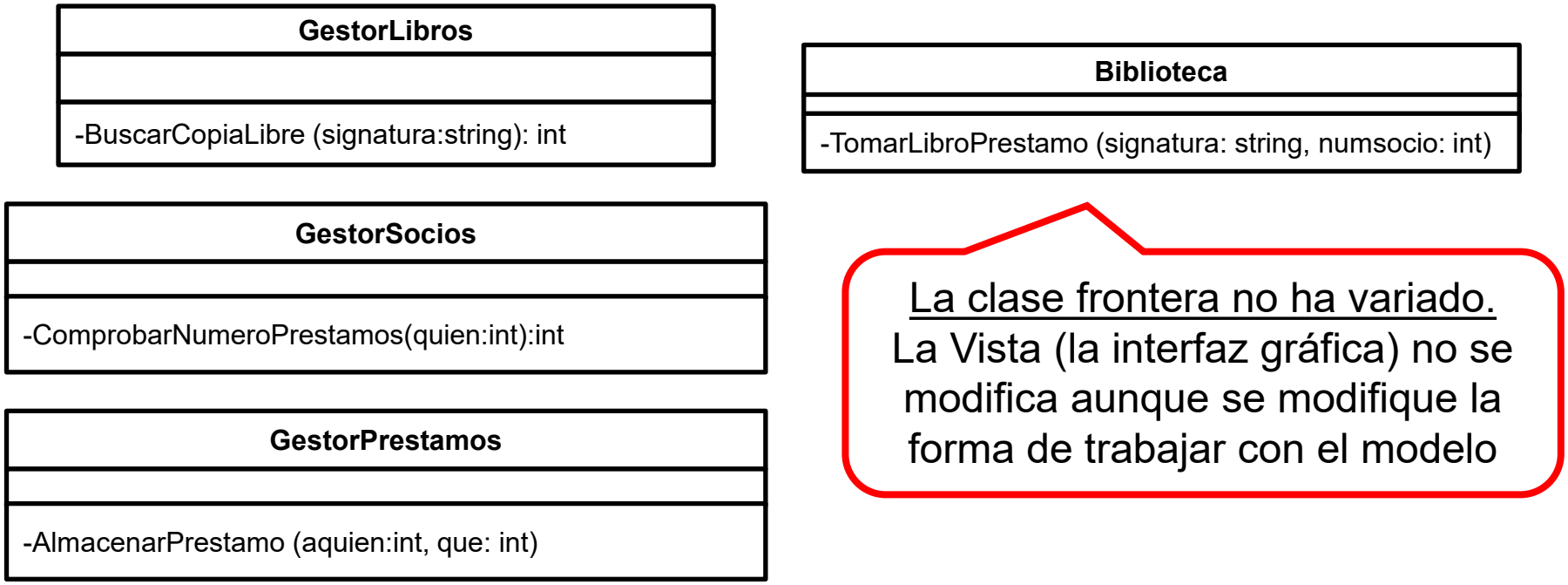
Diagrama de comunicación. Ejemplo

1: Introduce datos y pulsa “Tomar en Préstamo”
2: TomarLibroPrestamo (signatura,numsocio)
    2.1: elCodigoCopiaLibre=BuscarCopiaLibre (signatura)
        2.1.1: execSQL(“*SELECT CodCopiaLibro FROM Copia_Libro WHERE LibroSignatura= %signatura% AND Libre=1*”) : ResultadoSQL
            2.1.1.1: new ResultadoSQL()
        2.1.2: next()
        2.1.3: getInt (“CodCopiaLibro”): int
    2.2: Numprestamos= ComprobarNumeroPrestamos(numsocio)
        2.2.1: execSQL(“*SELECT COUNT(*) AS numprestamos FROM Prestamo WHERE SocioNumSocio= %numsocio%*) : ResultadoSQL
            2.2.1.1: new ResultadoSQL()
        2.2.2: next()
        2.2.3:getInt(“numprestamos”): int
    2.3: AlmacenarPrestamo (numsocio,elCodigoCopiaLibre)
        2.3.1: execSQL(“*INSERT INTO Prestamo (Copia_LibroCodCopiaLibro, SocioNumSocio) VALUES (%ElCodigoCopiaLibre%,%numsocio%,%fecha%)*”)
        2.3.2: execSQL(“*UPDATE Copia_Libro SET Libre=0 WHERE CodCopiaLibro= %elCodigoCopiaLibre%*”)

No generan ResultadoSQL

Diagrama de comunicación

- El diagrama de clases necesario para ese caso de uso sería ligeramente distinto



Arquitectura

- Descripción de la arquitectura en la fase de análisis
  - División del sistema en paquetes
    - Paquetes de servicio: agrupan clases cuyo objetivo es proporcionar servicios (ej: librerías externas)
    - Paquetes de entidad: agrupan las clases del dominio
    - Paquetes de interfaz: agrupan las clases relacionadas con la interfaz gráfica
    - Paquetes de control: agrupan las clases con la lógica del proceso