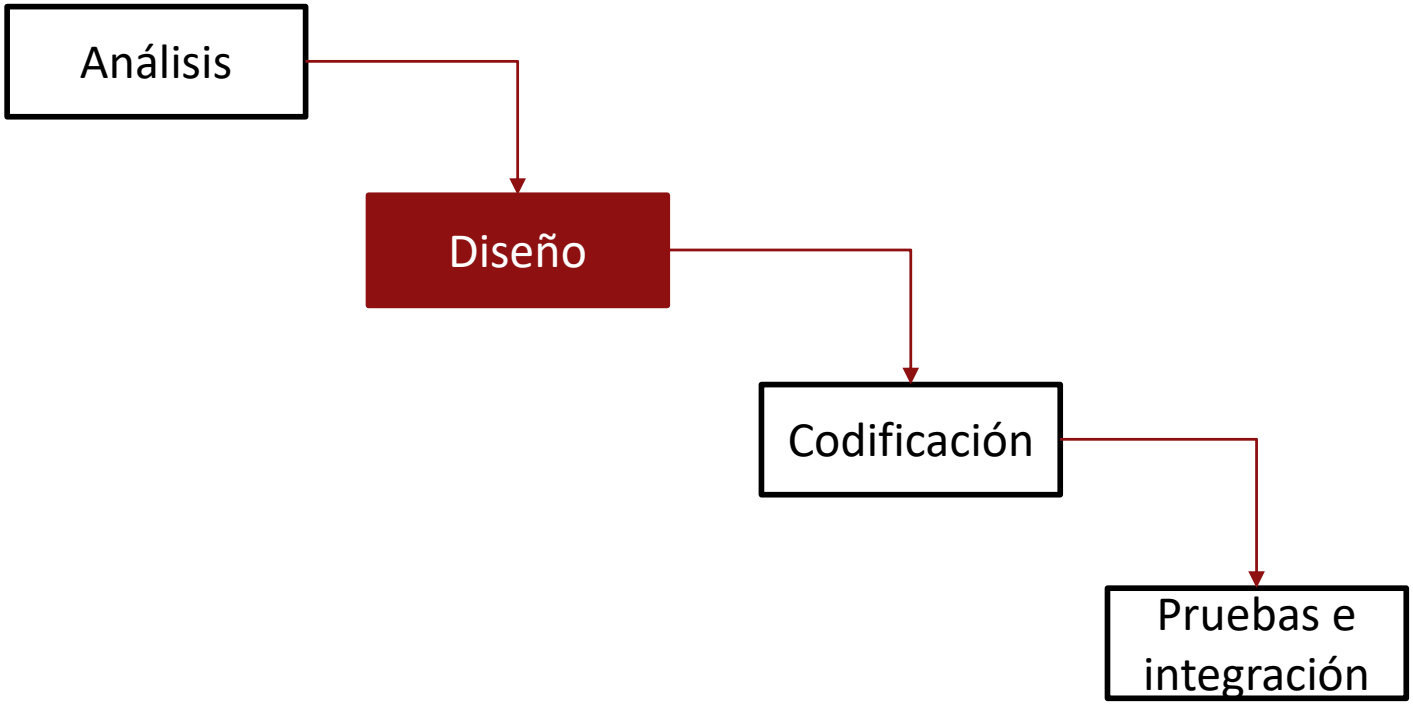
Metodología de la Programación

Fase de diseño

Universidad Rey Juan Carlos

Fase de diseño



Fase de diseño

- Es el proceso de definición de la arquitectura, componentes, módulos, interfaces, procedimientos de prueba y datos de un sistema software para satisfacer unos requisitos especificados.
- Se pasa del qué al cómo:
  - ¿Qué hay que hacer? → Especificación de requisitos.
  - ¿Cómo hay que hacerlo? → Especificaciones de diseño.

Fase de diseño

- Dentro de esta fase existen dos niveles:
  - Diseño de alto nivel.
  - Diseño de bajo nivel.
- La principal idea es ir detallando cada vez los aspectos del software que vamos a realizar.

- Para ello usaremos UML.

Unified Modeling Language (UML)

- A medida de los años 90 existían muchos métodos de análisis y diseño **OO**.
  - Mismos conceptos con distinta notación.
  - Mucha confusión.
  - “Guerra de los métodos”
- En 1994, Booch, Rumbaugh y Jacobson deciden unificar los métodos:

  Unified Modeling Language (UML)

Ventajas de UML

- Reunir los puntos fuertes de cada método
- Idear nuevas mejoras
- Proporcionar estabilidad en el mercado
  - Proyectos basados en un lenguaje maduro
  - Aparición de potentes herramientas
- Eliminar la confusión de los usuarios

Objetivos en el diseño de UML

- Modelar sistemas, desde los requisitos hasta los artefactos ejecutables, utilizando técnicas de OO.
- Cubrir las cuestiones relacionadas con el tamaño propias de los sistemas complejos y críticos.
- Lenguaje utilizable por personas y máquinas.
- Encontrar un equilibrio entre expresividad y simplicidad.

UML y modelado

- UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos (modelos) de un sistema que involucra una gran cantidad de software, desde una perspectiva OO.

¿Por qué modelados?

- “Una empresa software con éxito es aquella que produce de manera consistente software de calidad que satisface las necesidades de los usuarios”

- “El modelado es la parte esencial de todas las actividades que conducen a la producción de software de calidad”

¿Por qué modelamos?

- “Un modelo es una especificación de la realidad”
- Construimos modelos para comprender mejor el sistema que estamos desarrollando.
- Cuatro utilidades de los modelos:
  - Visualizar cómo es o queremos que sea el sistema.
  - Especificar la estructura y comportamiento del sistema.
  - Proporcionan plantillas que guían la construcción del sistema.
  - Documentan las decisiones.

¿Por qué las empresas no modelan?

- La mayor parte de las empresas software no realizan ningún modelado.
- El modelado requiere:
  - Aplicar un proceso de desarrollo.
  - Formación del equipo en las técnicas.
  - Tiempo
- ¿Se obtienen beneficios del modelado?

Diseño UML

- En el paradigma estructurado, el bloque fundamental es la operación.
  - Funciones y datos van por separado.

- El paradigma de orientación a objetos presta igual atención a datos y operaciones.
  - Objetos: información del estado y operaciones asociadas en una sola unidad.

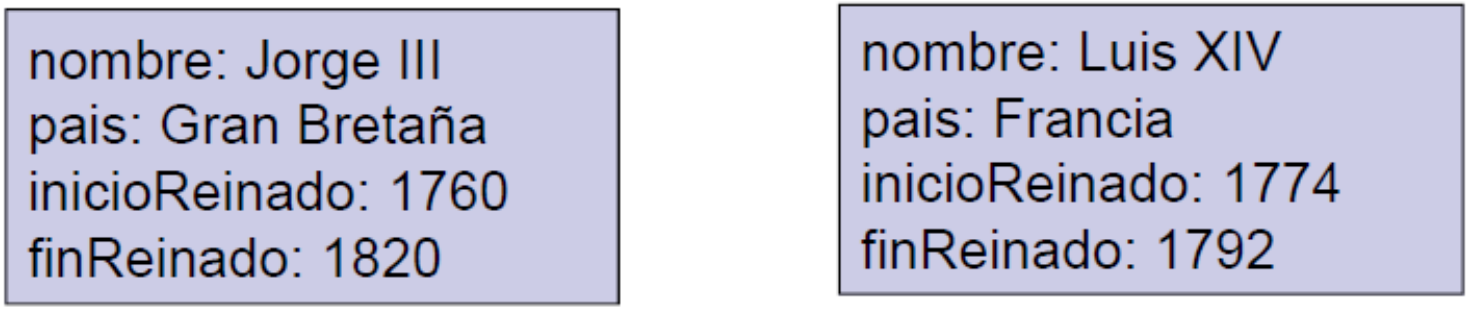
Diseño UML

- Diferentes tipos de diagramas:
  - **Modelado de la estructura**:
    - Diagrama de clases.
  - Modelado de comportamiento:
    - Diagrama de Transición de Estados.
    - Diagrama de Secuencia.

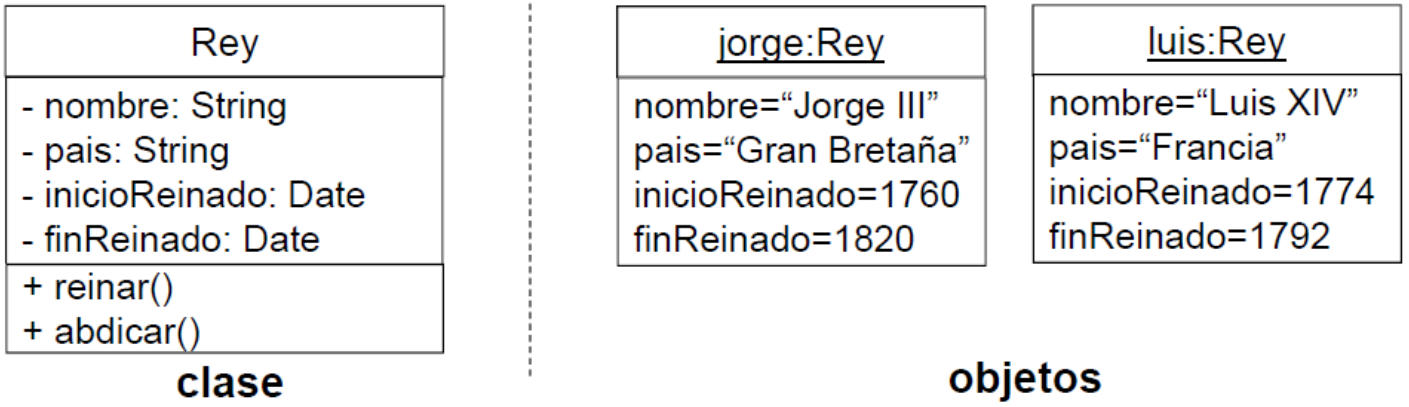
- Matriz de Trazabilidad

Diseño UML

- Clases y objetos



- Todos los reyes tienen aspectos en común.
- Podemos representar esos aspectos en una clase.
- Reyes concretos serían instancias (objetos) de dicha clase.



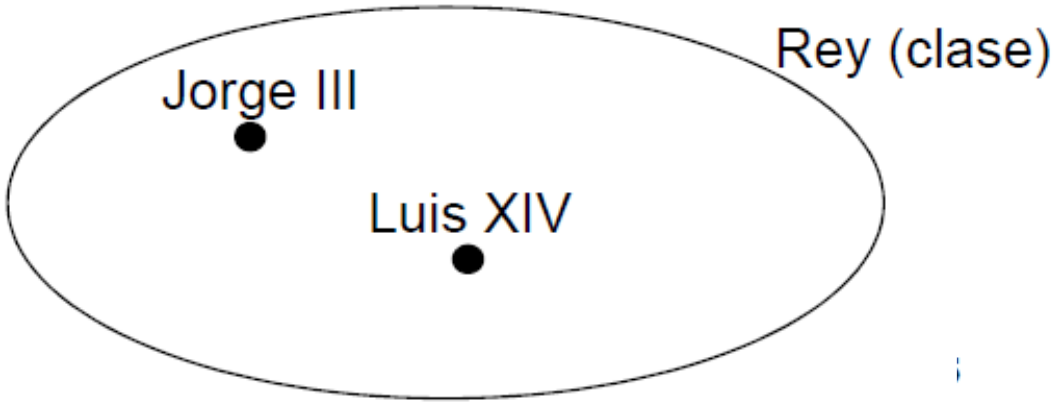
Diseño UML

Clase

- Declara propiedades (atributos) de todas las instancias.
- Declara operaciones (métodos) aplicables a todas las instancias.

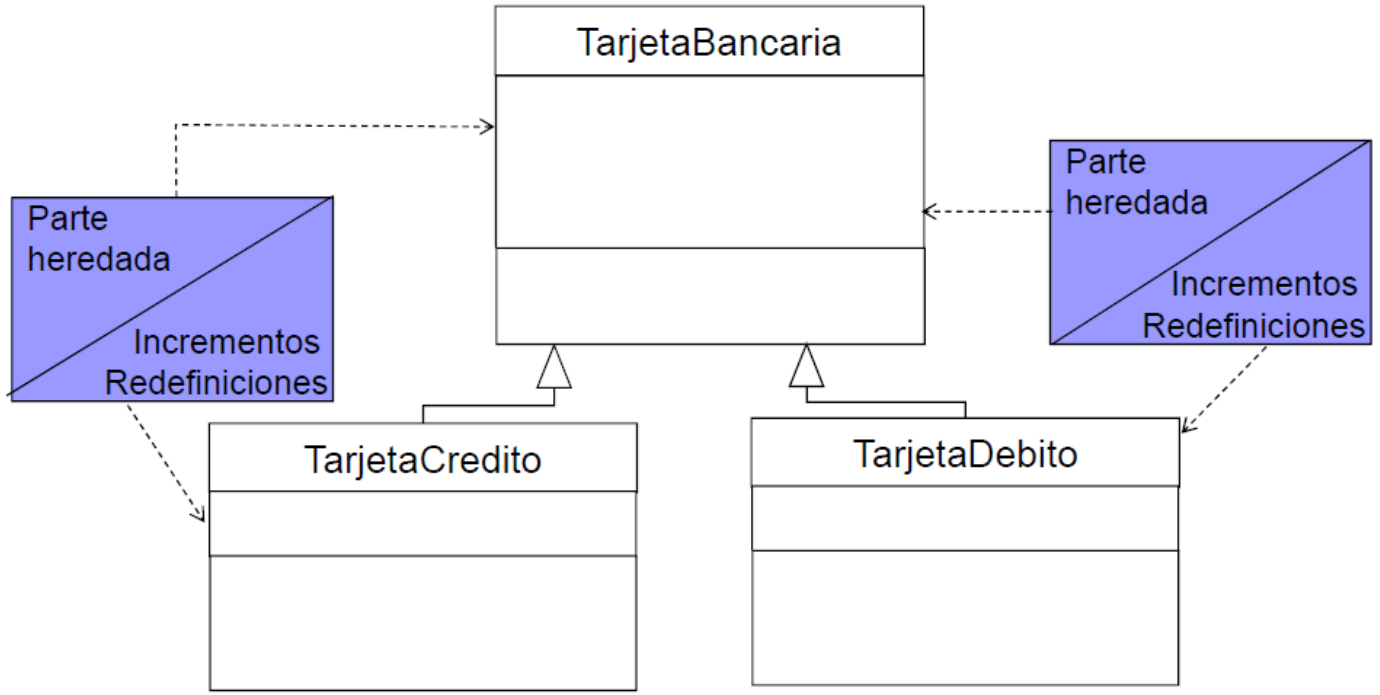
Objeto

- Da valores a los atributos declarados en la clase.
- Reacciona a invocaciones de los métodos declarados en la clase, usando como estado los valores de sus atributos.



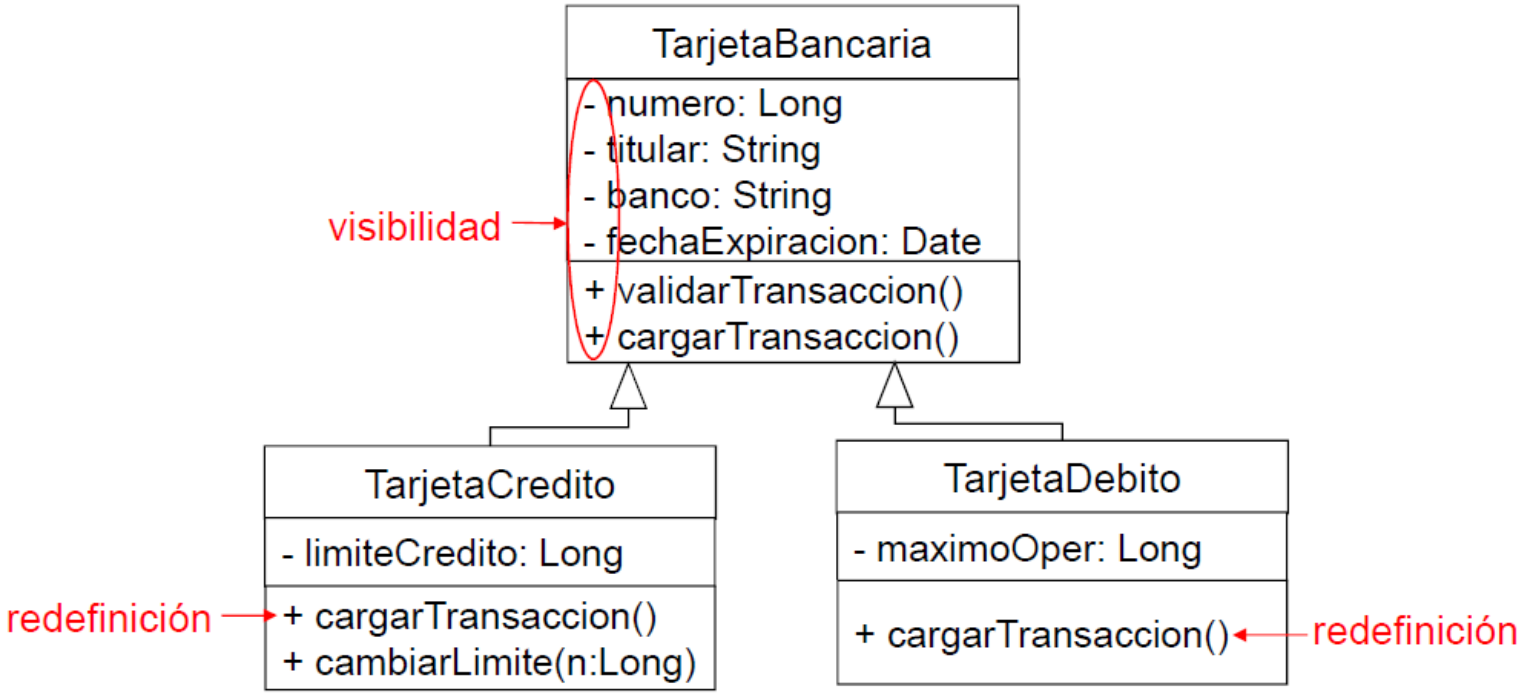
Diseño UML (Herencia)

- Las herencias son jerarquías de especialización de clases.
- Herencia de propiedades y operaciones.



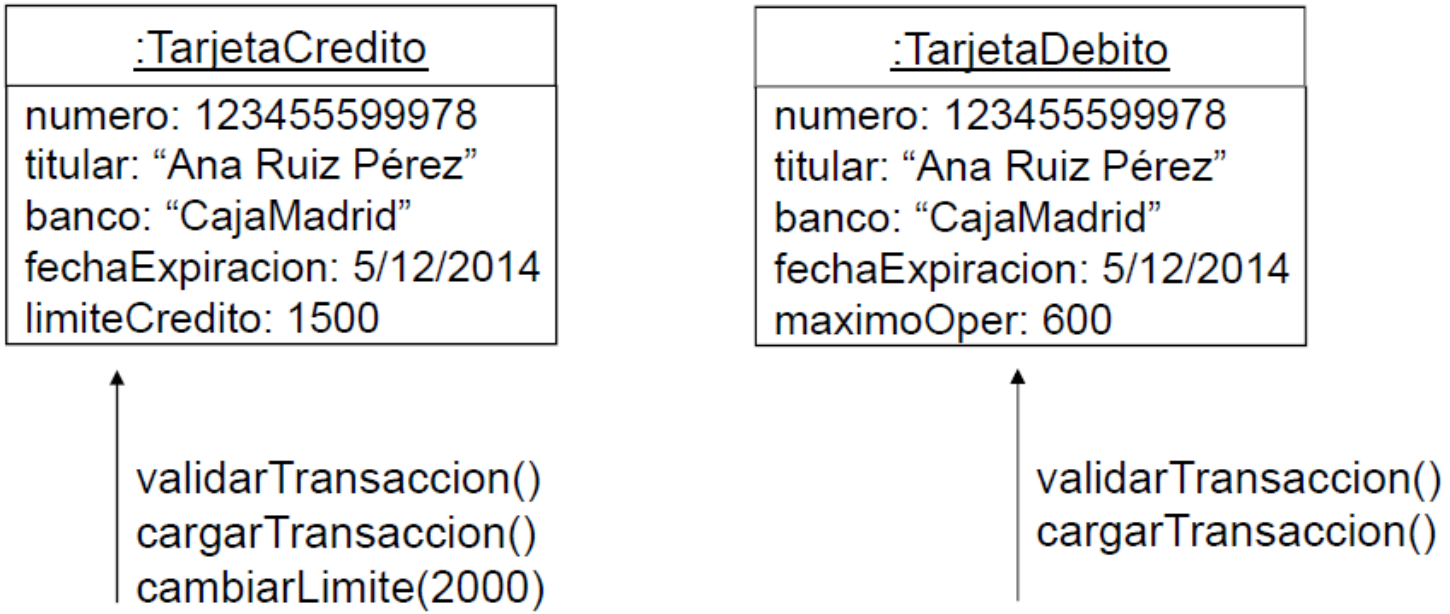
Diseño UML (Herencia)

- Las herencias son jerarquías de especialización de clases.
- Herencia de propiedades y operaciones.



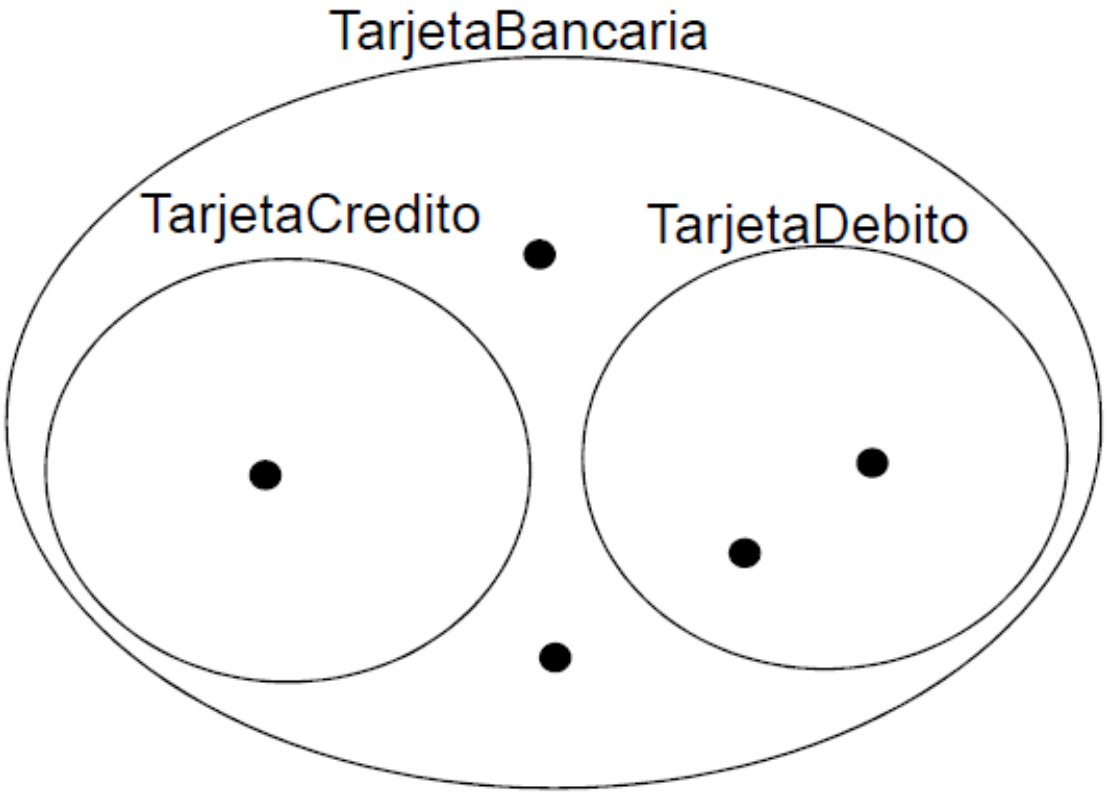
Diseño UML (Herencia)

- Un objeto de una clase hija hereda las propiedades de la clase padre.
- Podemos invocarle las operaciones definidas en la clase padre.



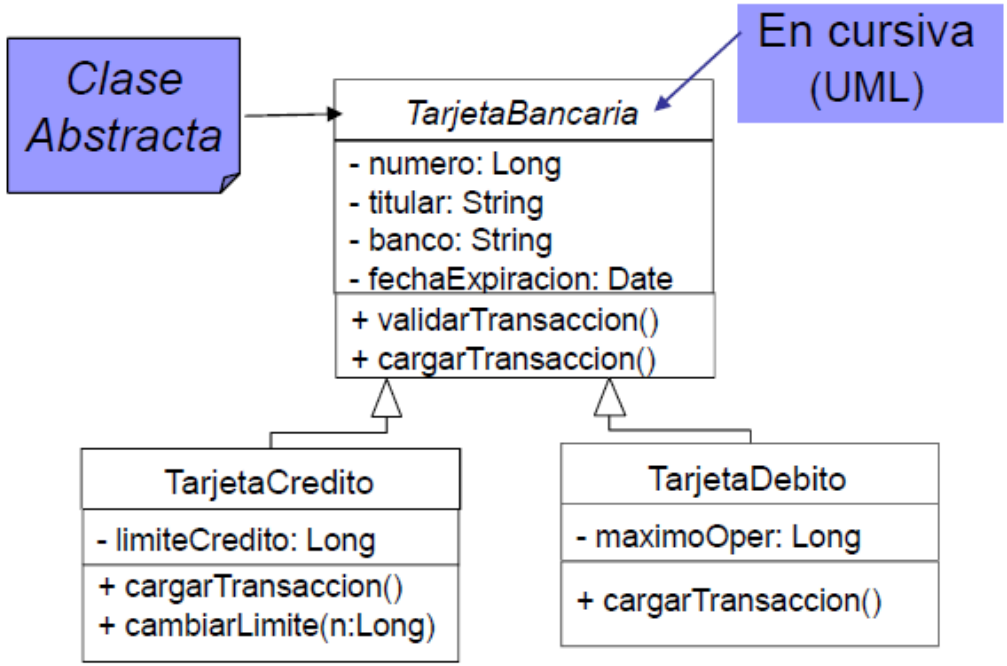
Diseño UML (Herencia)

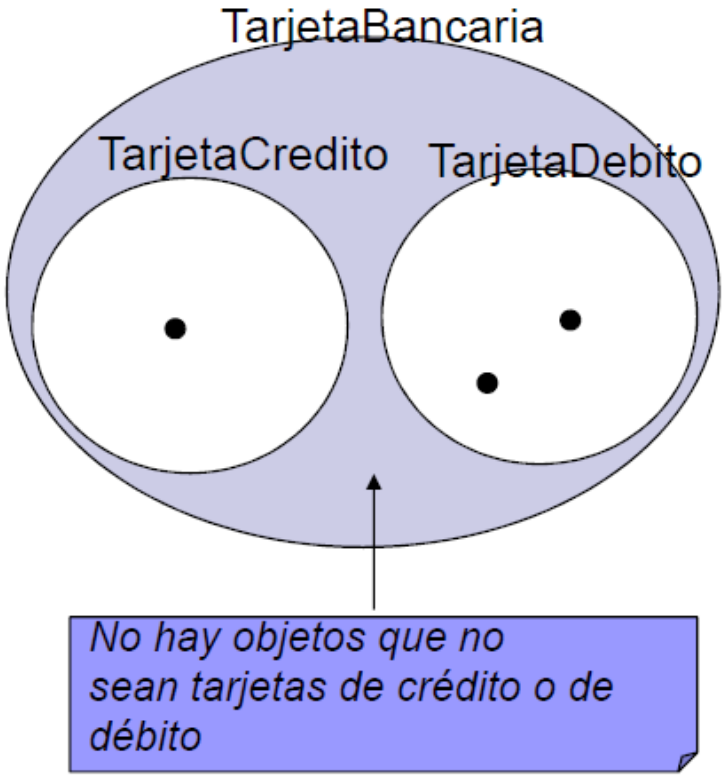
- La substitución segura de supertipos por subtipos.
  - Todas las tarjetas de crédito y de débito son tarjetas bancarias
  - Existen tarjetas bancarias que no son de crédito ni de débito.
  - No hay tarjetas que sean de crédito y de débito a la vez.



Diseño UML (Clases Abstractas)

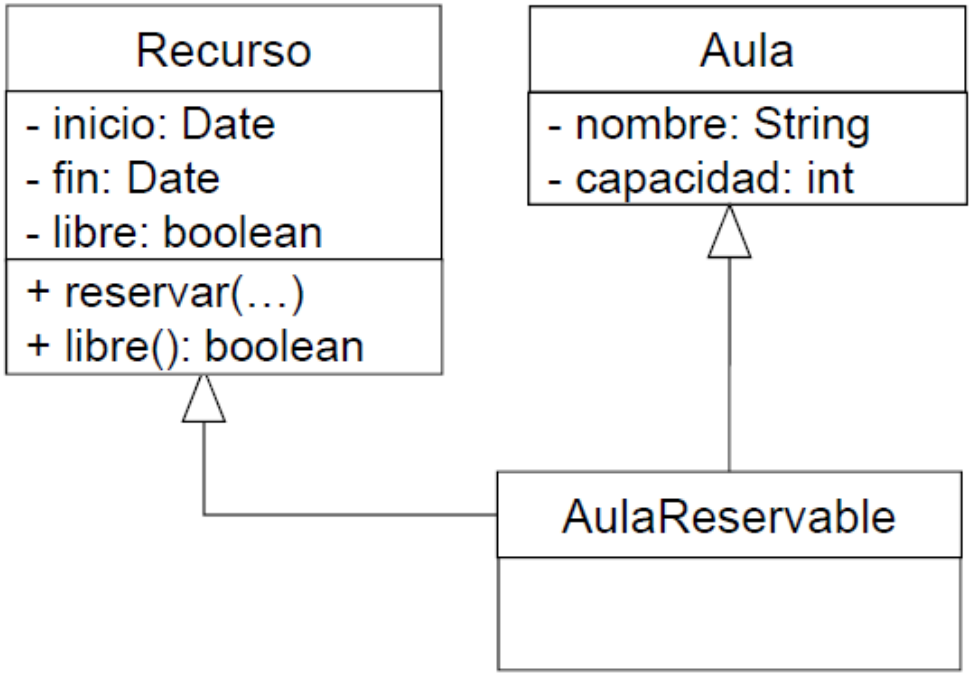
- Una clase abstracta no puede instanciarse

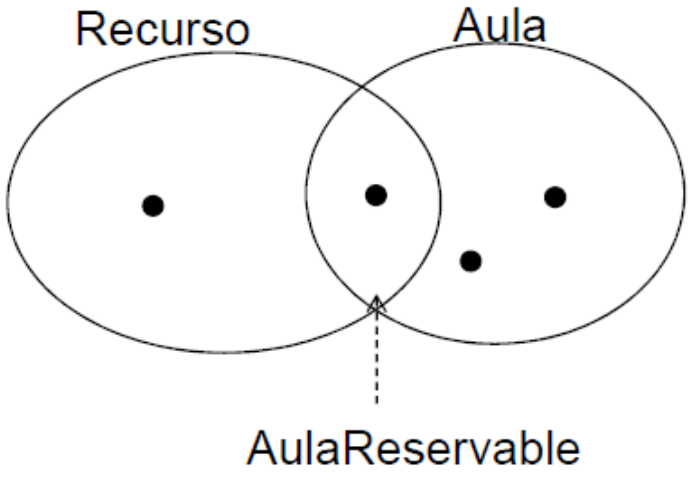




Diseño UML (Herencia Múltiple)

- Una clase puede heredar de varias.
- No es posible en Java, pero sí en otros lenguajes de programación como C++.



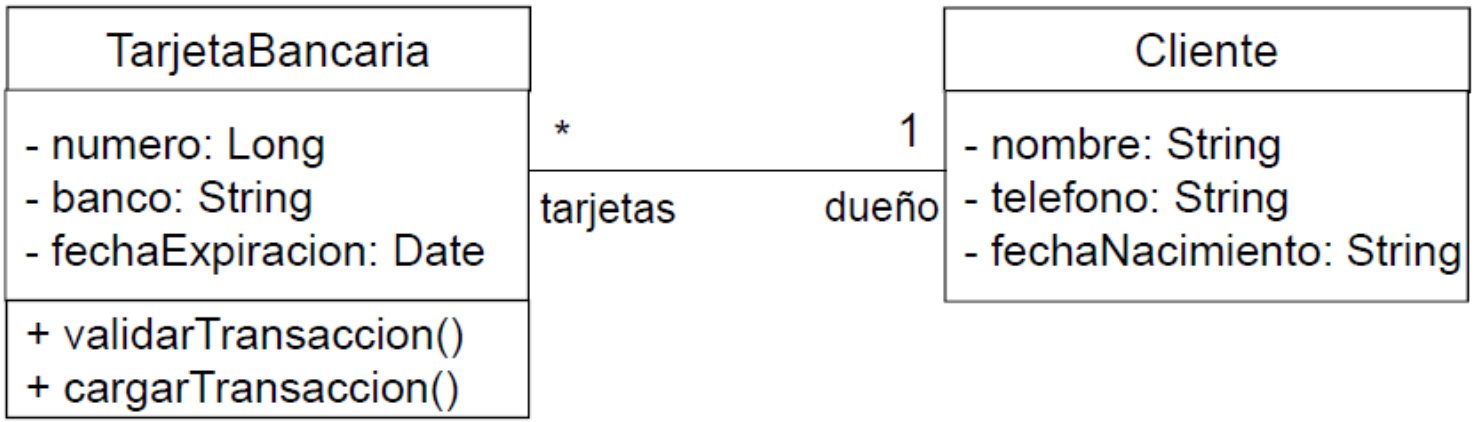


Diseño UML (Asociaciones)

- Los objetos no están aislados, si no que necesitan "conocerse" para poder invocar métodos de otros objetos.
  - Esta funcionalidad se implementa mediante colaboraciones de objetos.
- Conceptualmente, se representa mediante un enlace (una línea) entre dos clases.
  - En programación, esto significa que un objeto tiene una referencia a otro objeto (un atributo).

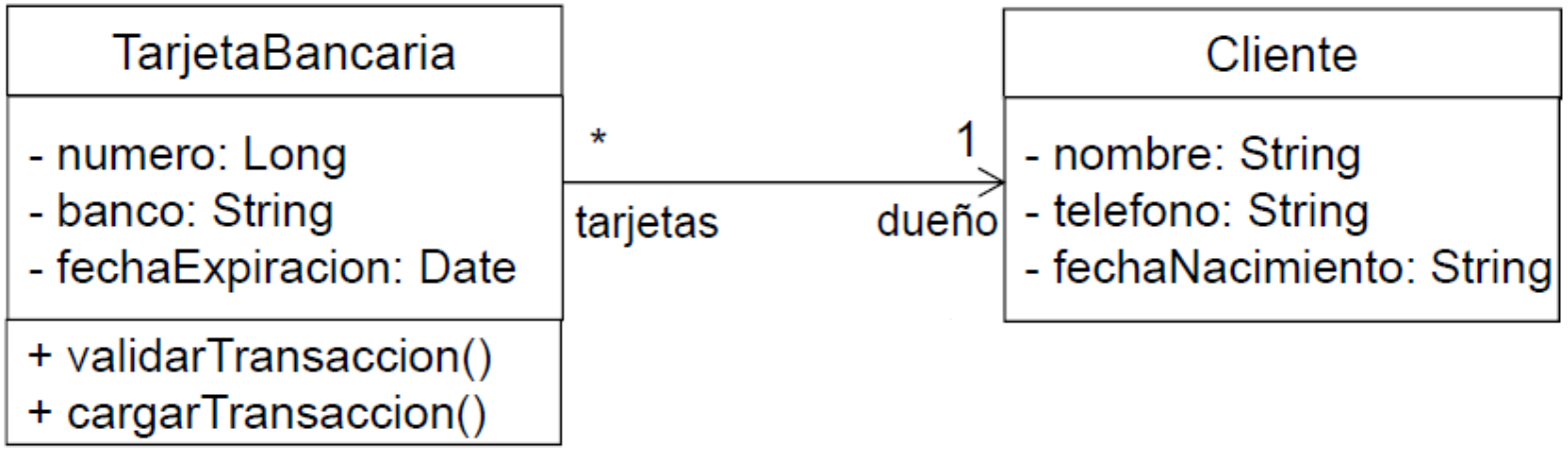
Diseño UML (Asociaciones)

- Multiplicidad:
  - Con cuántos objetos de destino se puede relacionar un objeto fuente, y viceversa.
- Roles:
  - Nombres en los extremos de las asociaciones.



Diseño UML (Asociaciones)

- Navegación:
  - Indica si un objeto puede acceder al otro.



Diseño UML (Asociaciones)

- Herencia de asociaciones:
  - Una asociación declarada en una clase base, se hereda en las clases hijas…

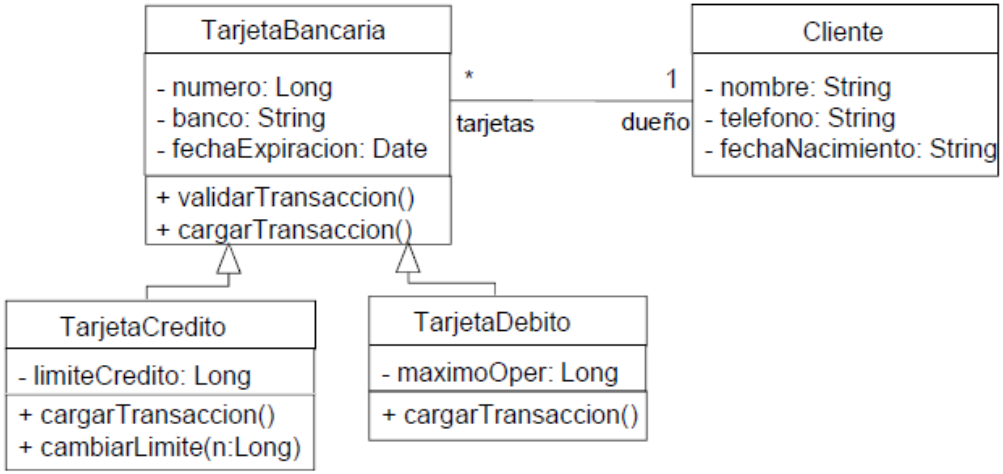


Diagrama de Clases

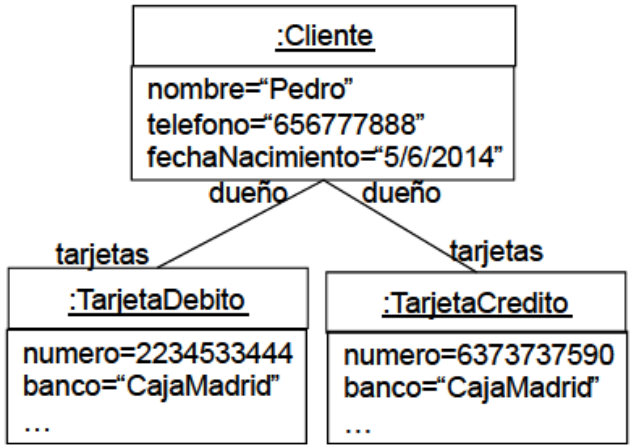
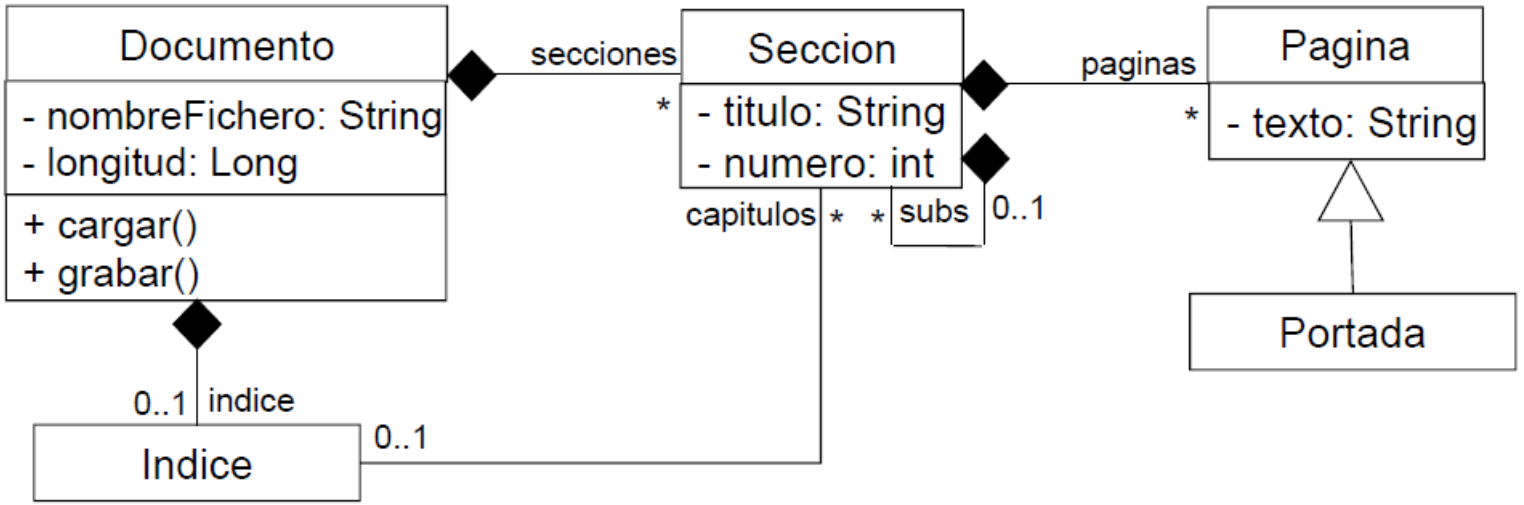


Diagrama de Objetos

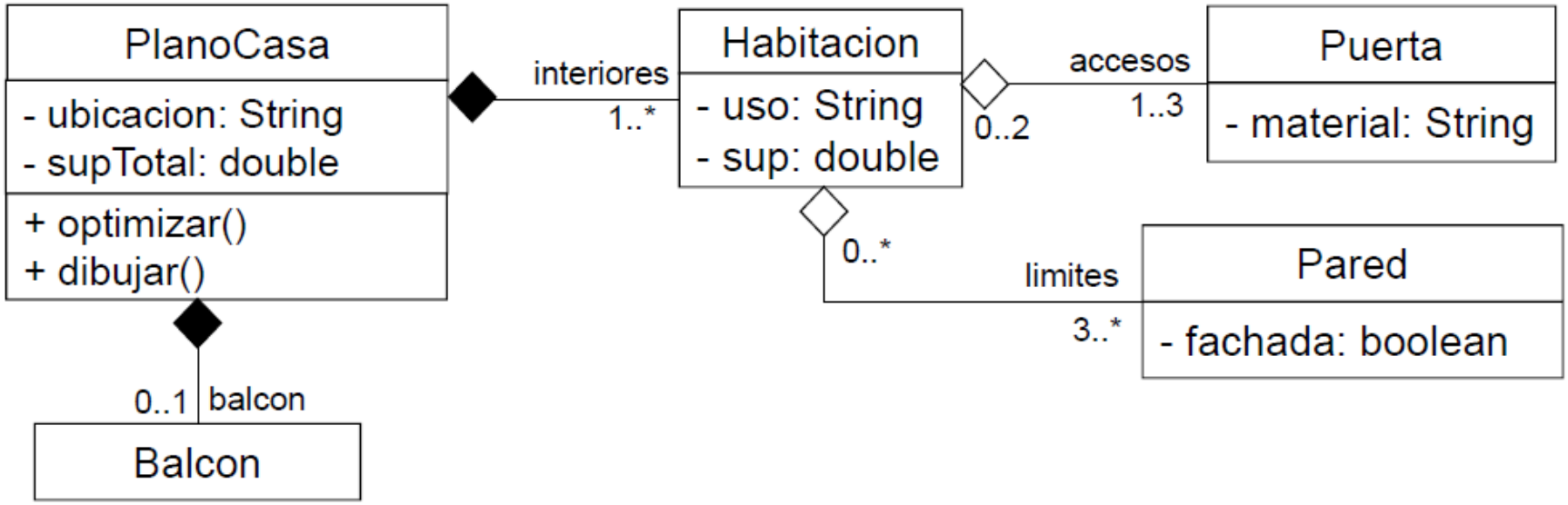
Diseño UML (Composición y Agregación)

- Hay relaciones con semánticas especiales:
  - ◆ Composición (una hecha de partes)
  - ◇ Agregación (composición débil)



Diseño UML (Composición y Agregación)

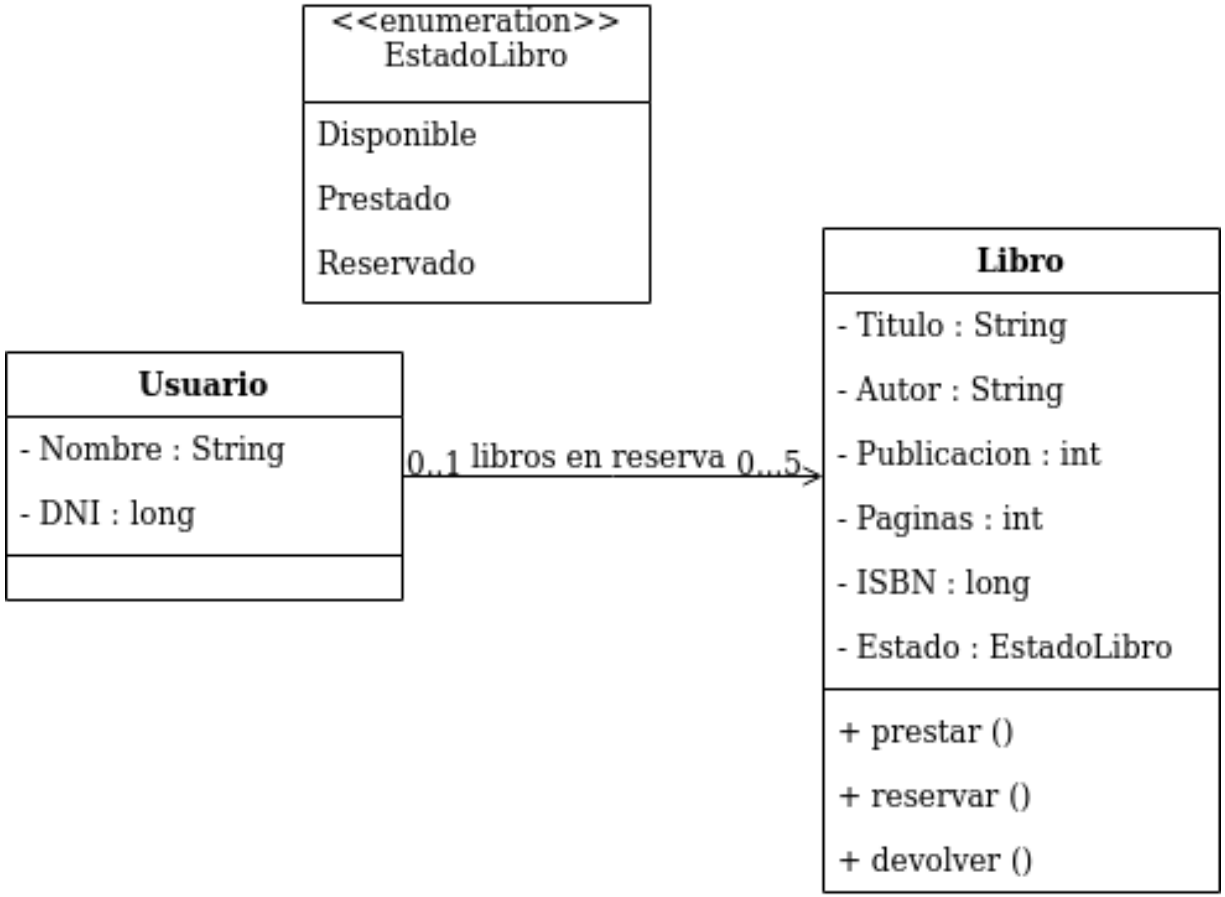
- Hay relaciones con semánticas especiales:
  - ◆ Composición (partes en solo un compuesto)
  - ◇ Agregación (partes pueden estar en varios compuestos)



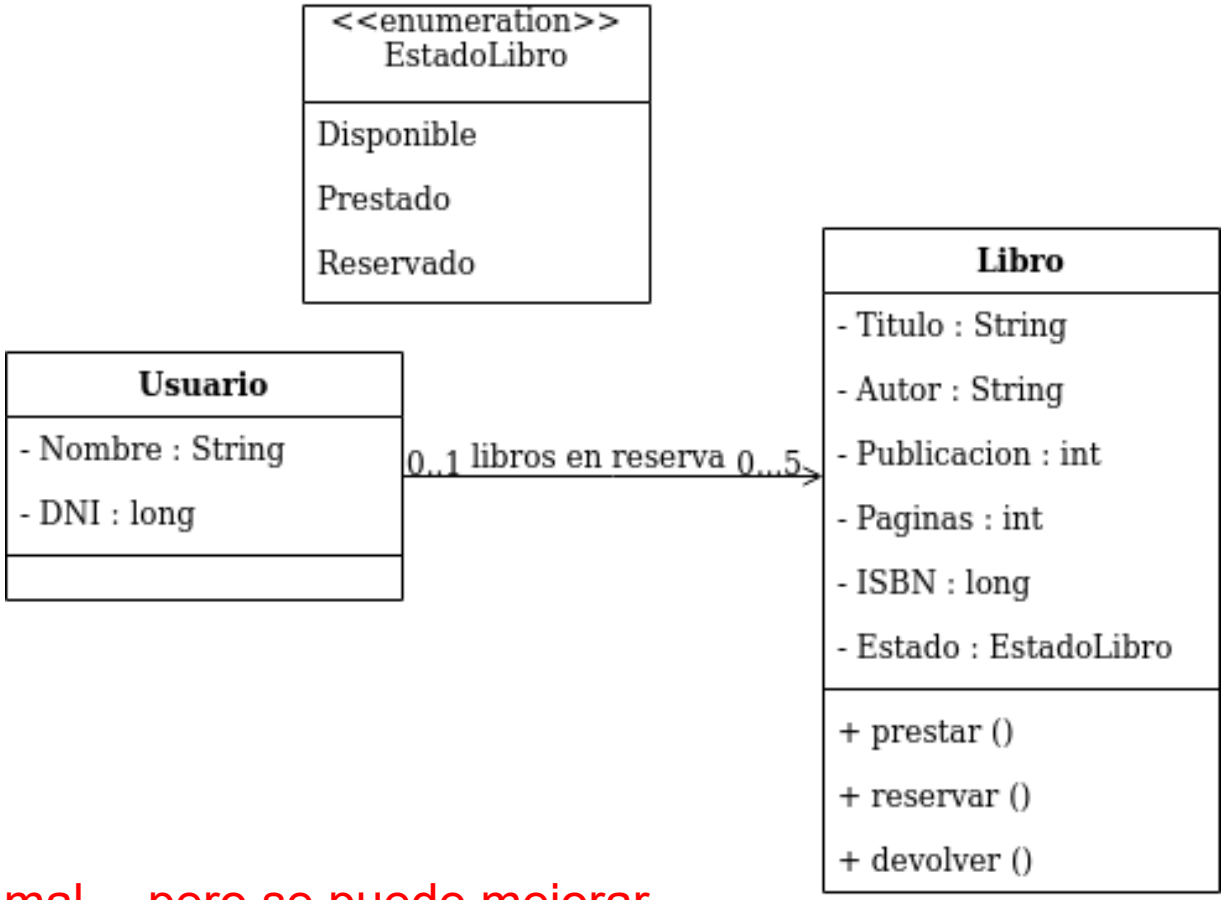
Ejercicio Diagrama de Clases

- Se quiere modelar el servicio de una biblioteca. La biblioteca tiene un conjunto de libros que tienen un título, un autor, año de publicación, número de páginas, e isbn.
- Los usuarios podrán tener prestados como máximo 5 libros.
- Los usuarios tienen un nombre y el DNI.
- Además, si un libro estuviera prestado, un usuario podría reservarlo (siempre y cuando no estuviera previamente reservado).
- El usuario que realiza la reserva, es el único que puede tomar prestado dicho libro, o cancelar la reserva.

Ejercicio Diagrama de Clases

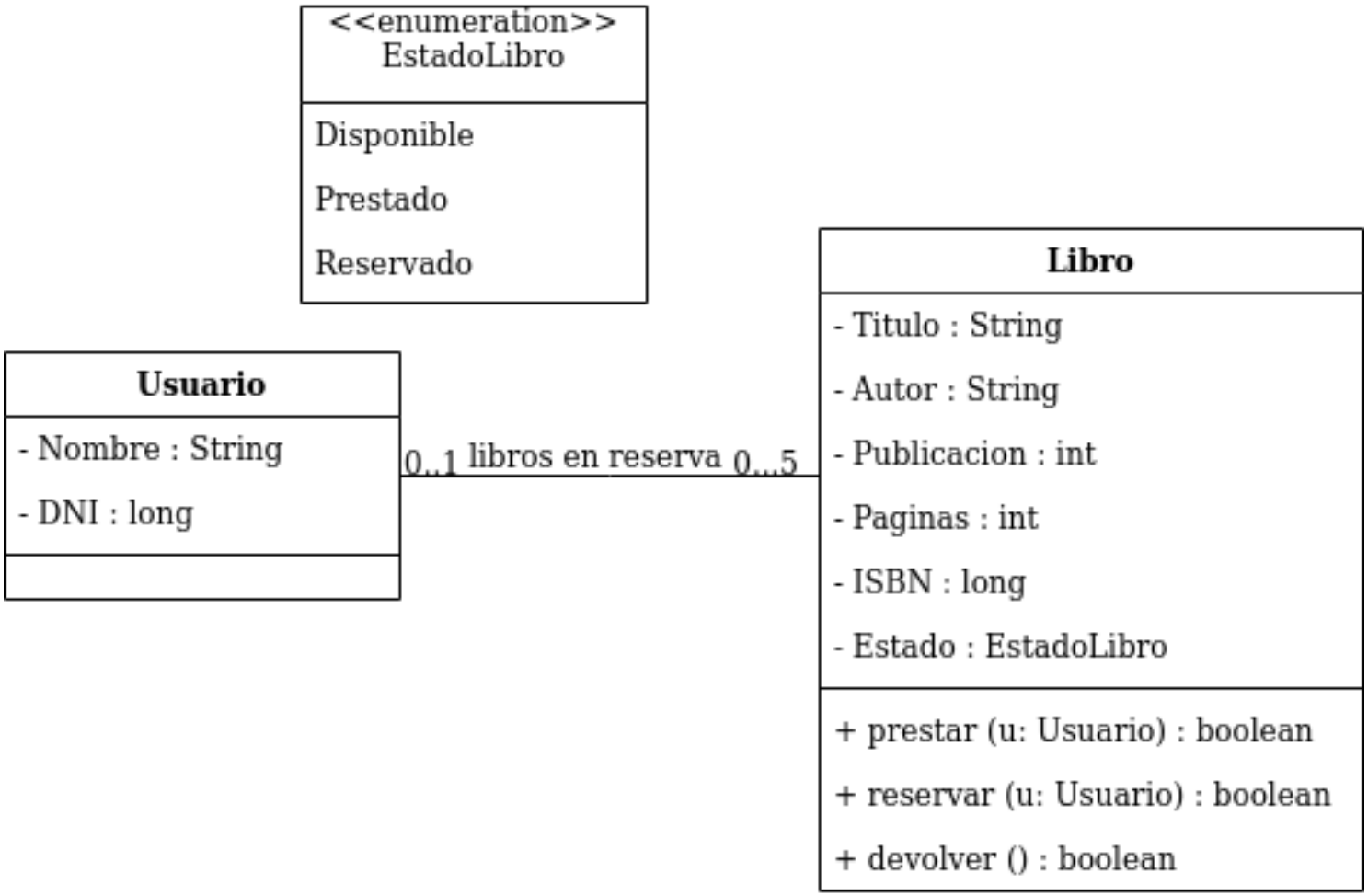


Ejercicio Diagrama de Clases



… no está mal… pero se puede mejorar

Ejercicio Diagrama de Clases

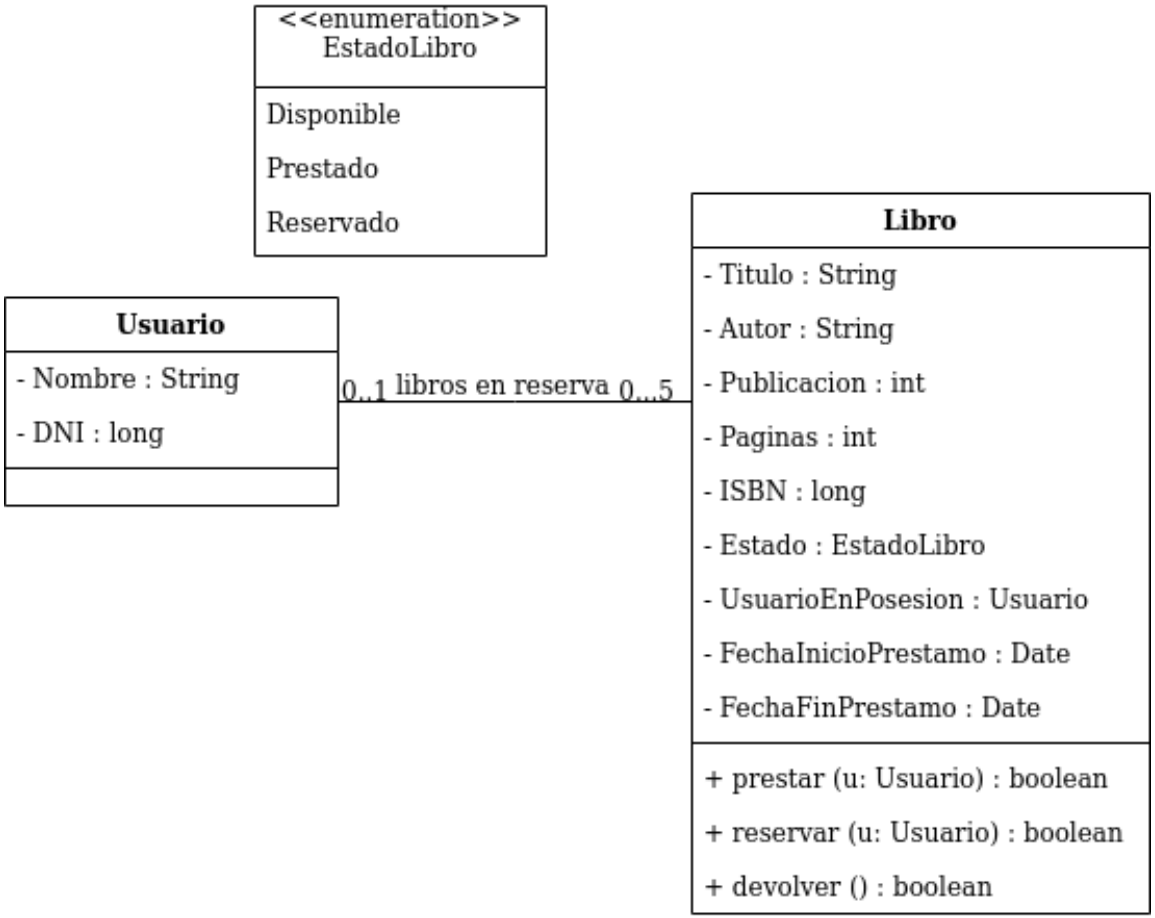


Ejercicio Diagrama de Clases (II)

- Cuando un usuario toma prestado un libro se almacena en el sistema la fecha en la que se ha efectuado, y se calcula la fecha de devolución (un mes mas tarde).
- En caso de que la devolución se produzca fuera del plazo permitido, el usuario será penalizado.
- Dicha penalización consiste en que el usuario no podrá tomar prestados libros de la biblioteca durante un mes.

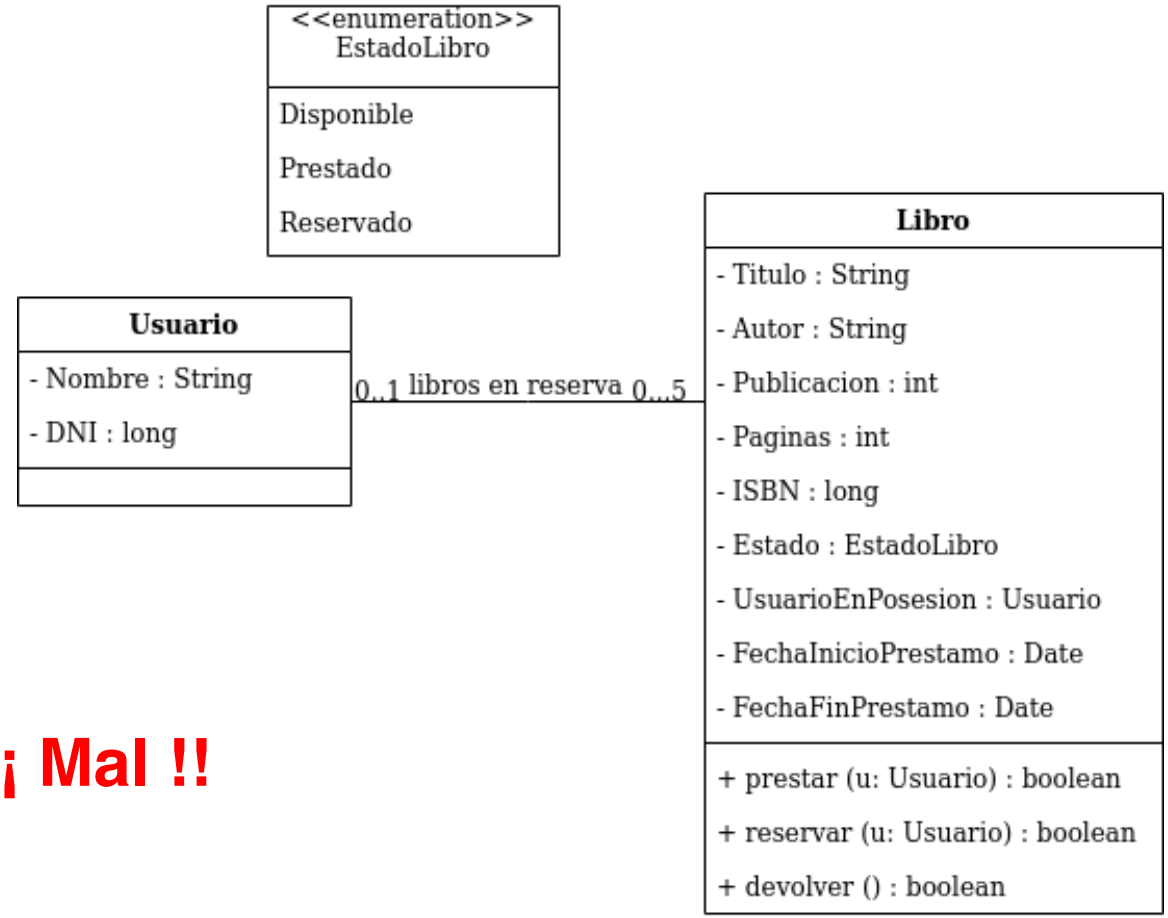
Ejercicio Diagrama de Clases (II)

- Incluimos las fechas de los préstamos…



Ejercicio Diagrama de Clases (II)

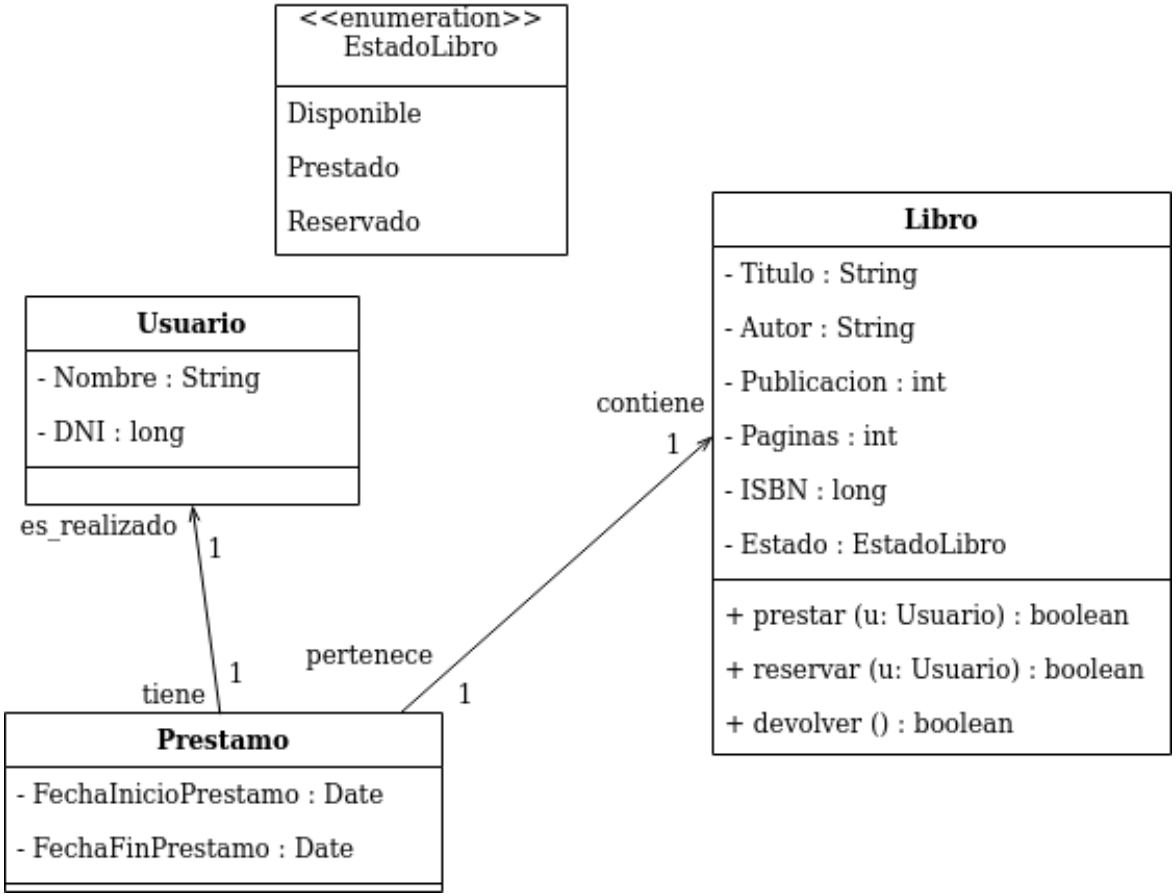
- Incluimos las fechas de los préstamos…



¡¡ Mal !!

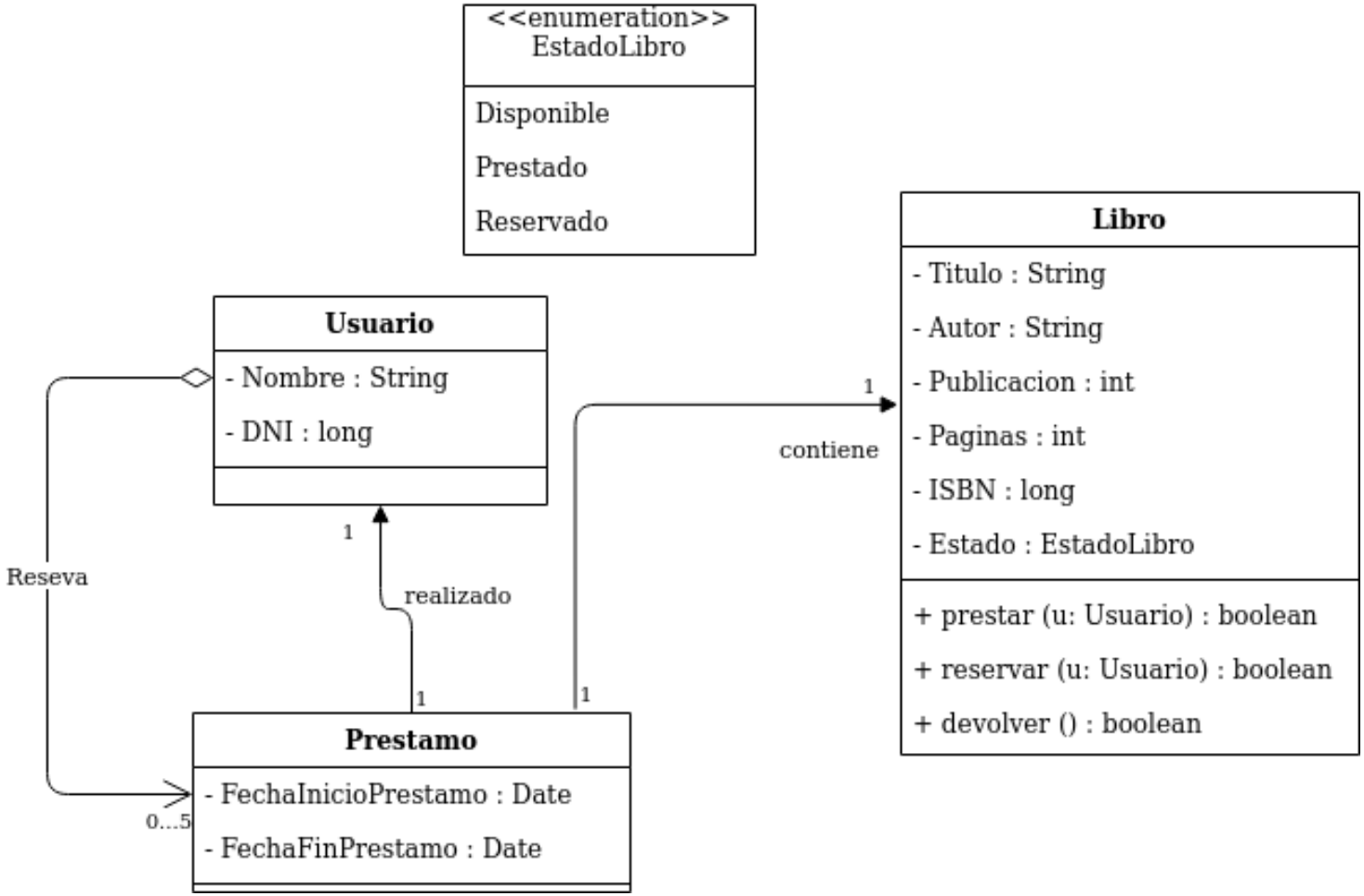
Ejercicio Diagrama de Clases (II)

- Incluimos las fechas de los préstamos…



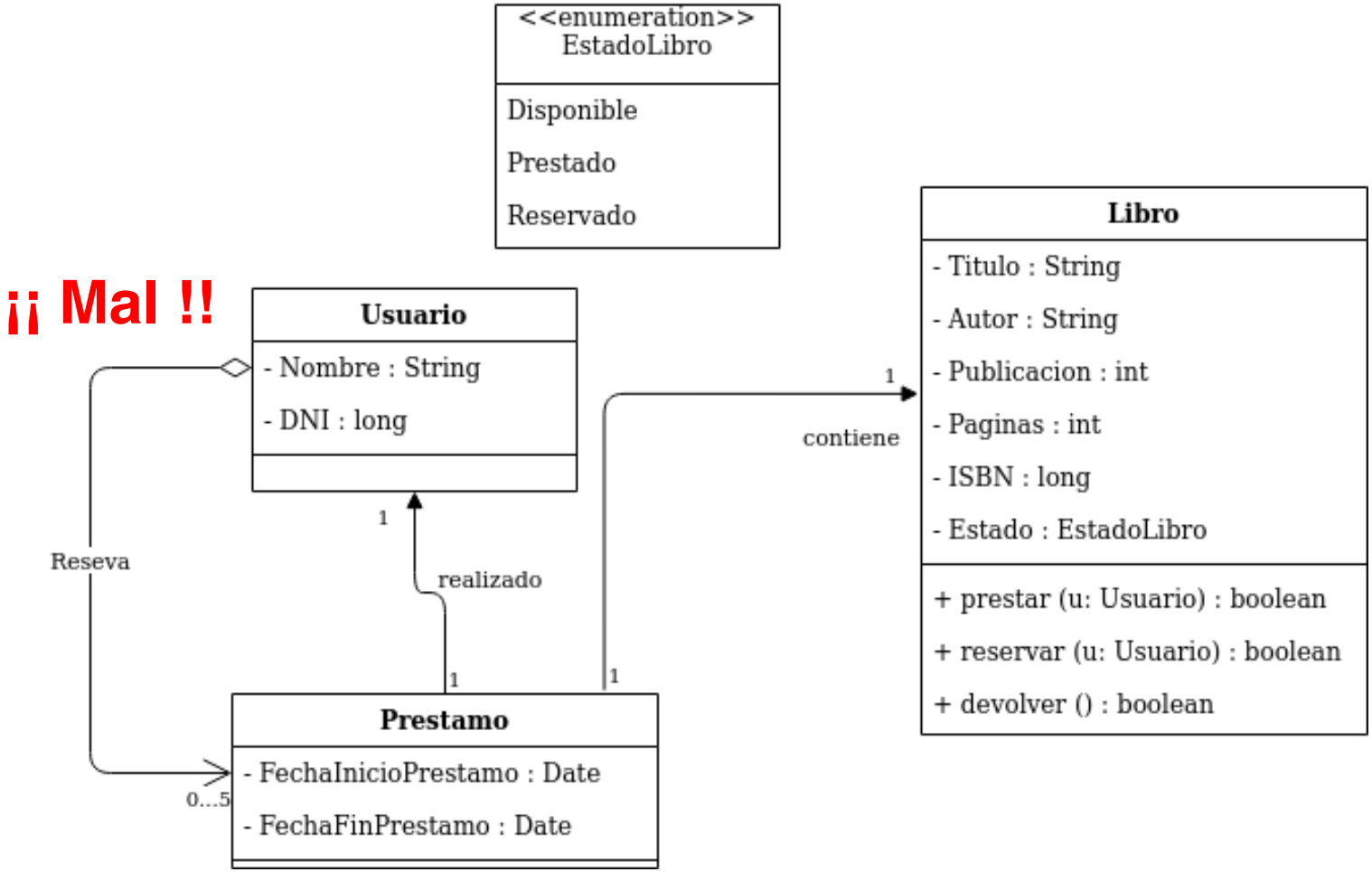
Ejercicio Diagrama de Clases (II)

- Añadimos la restricción de 5 préstamos



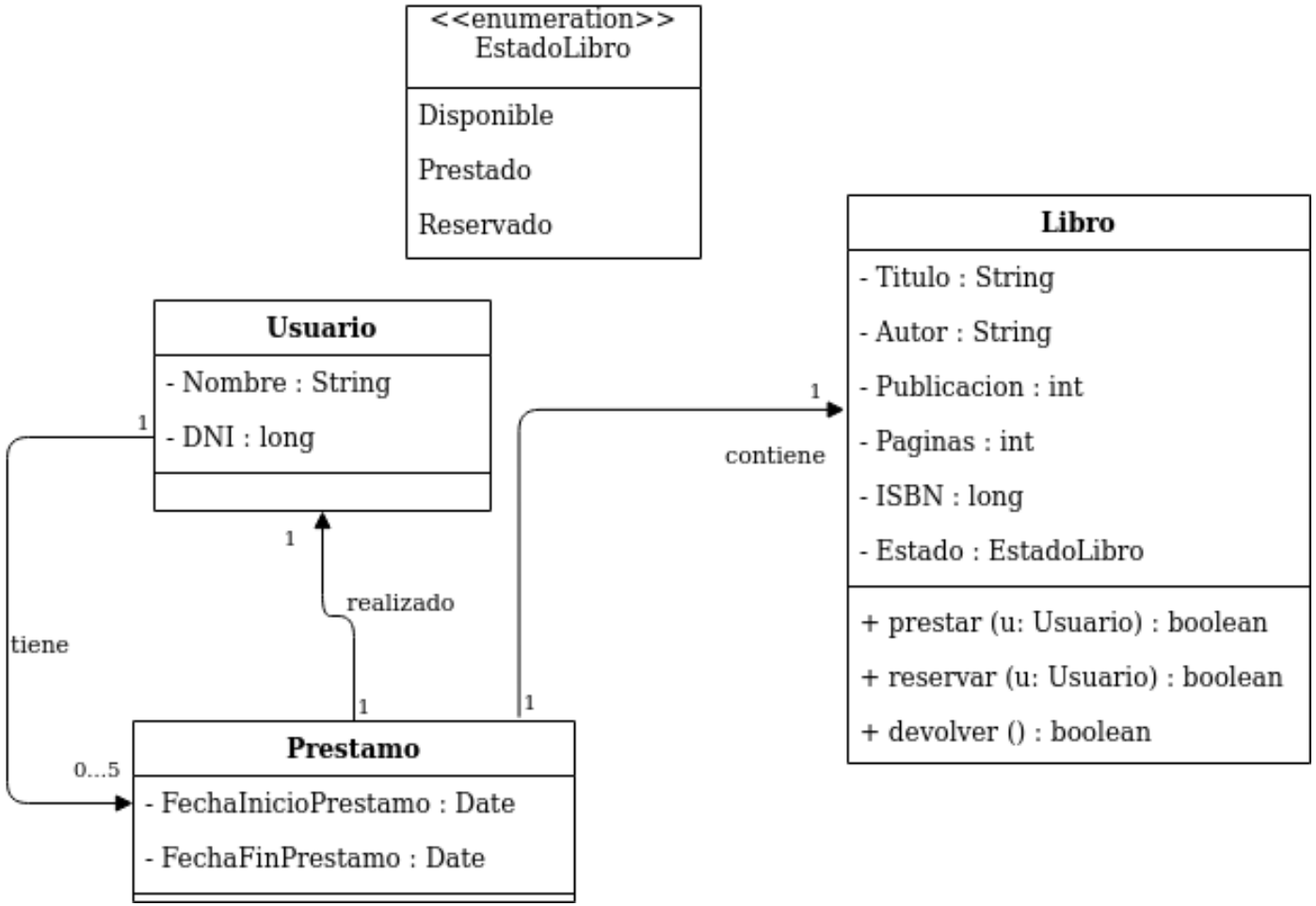
Ejercicio Diagrama de Clases (II)

- Añadimos la restricción de 5 préstamos



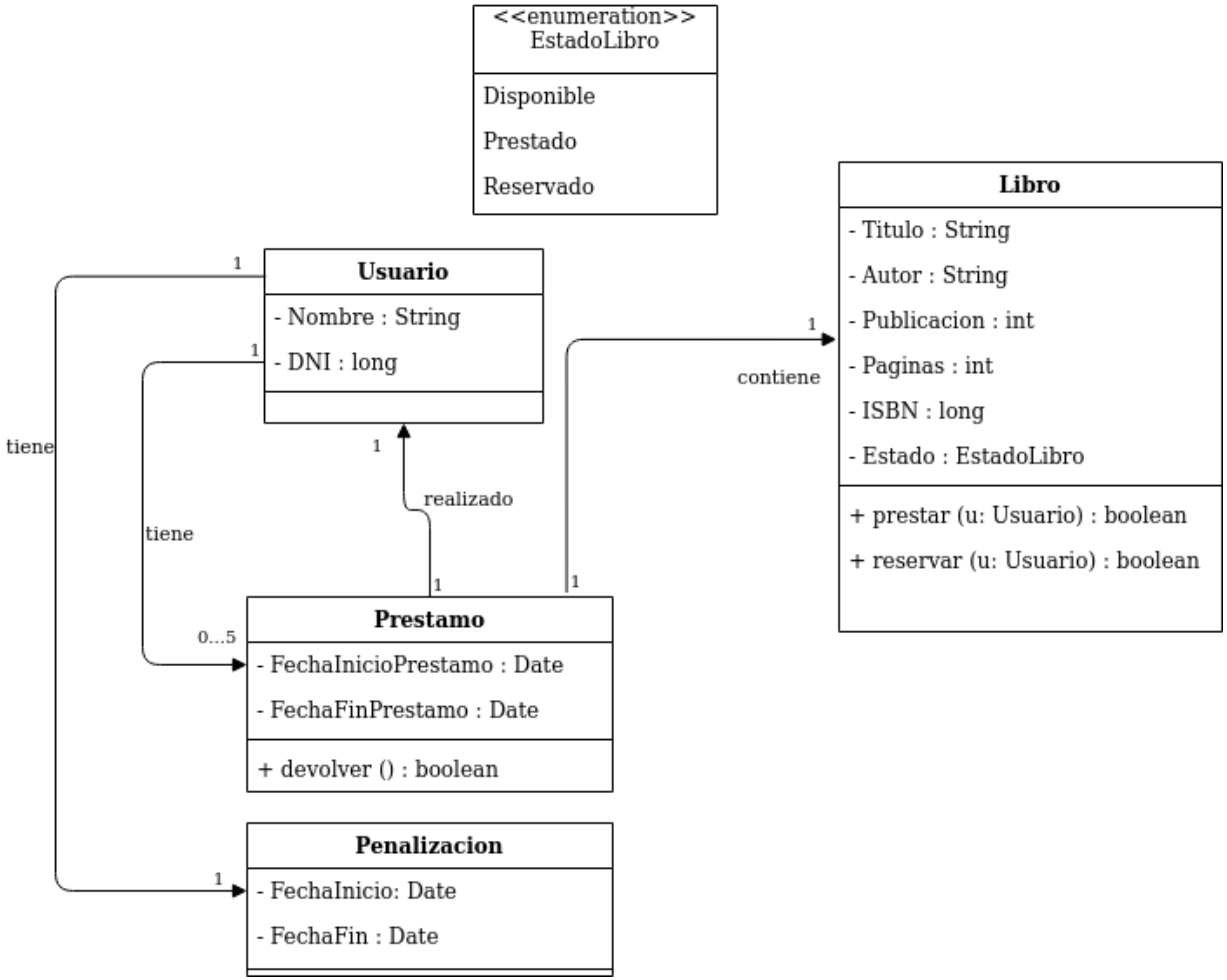
Ejercicio Diagrama de Clases (II)

- Añadimos la restricción de 5 préstamos



Ejercicio Diagrama de Clases (II)

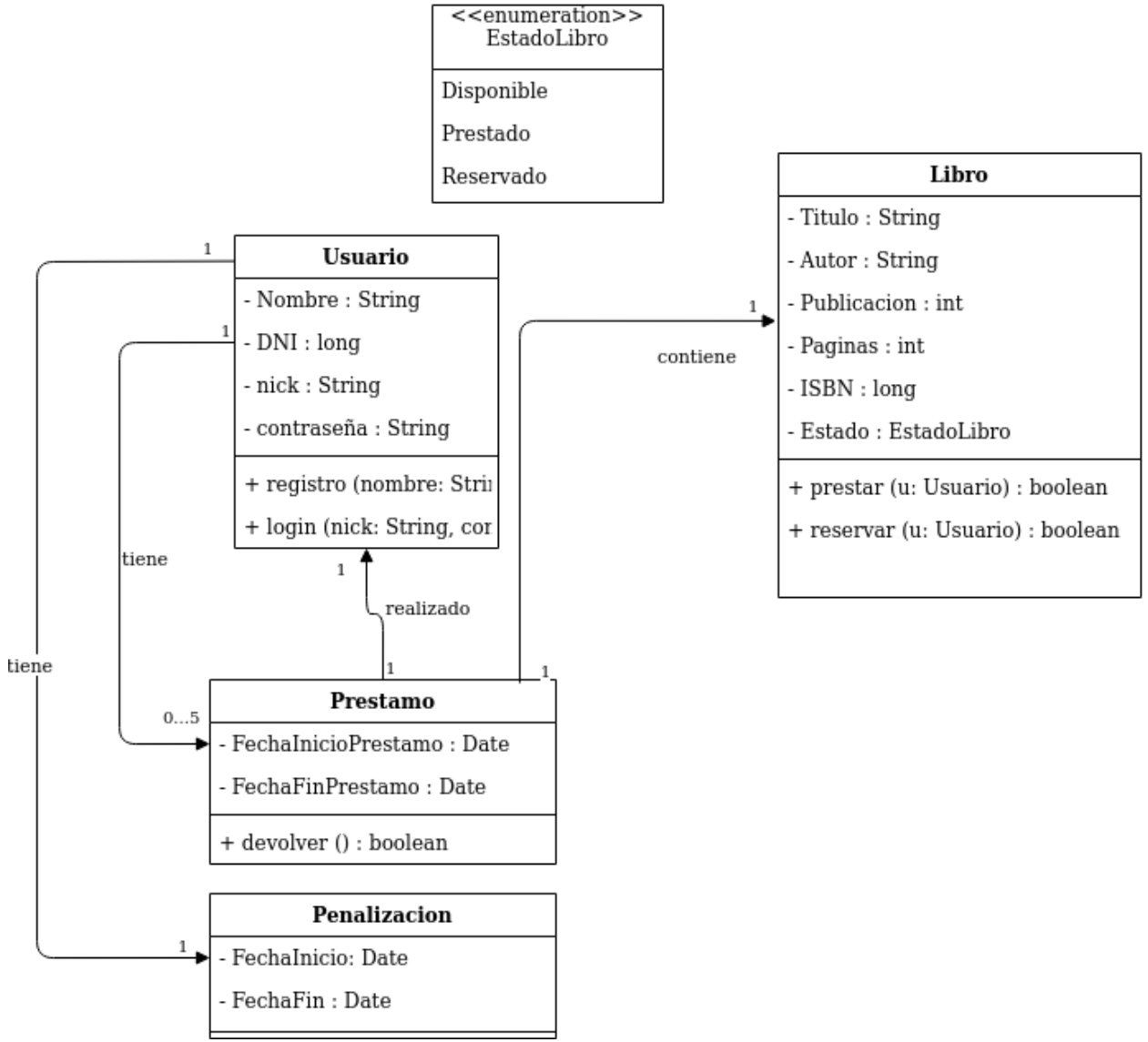
- Añadimos las penalizaciones



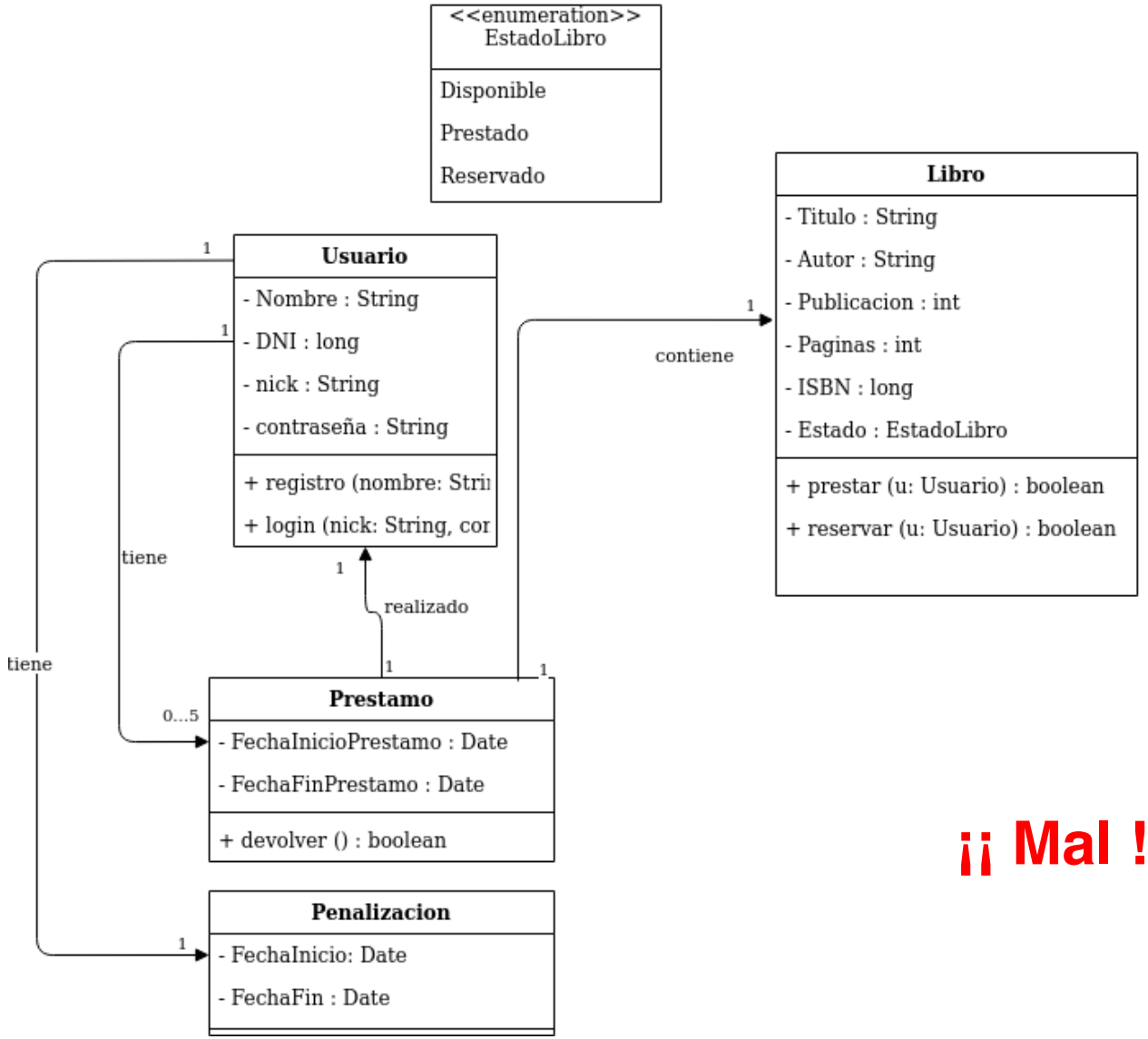
Ejercicio Diagrama de Clases (III)

- Los usuarios pueden acceder al sistema con un nick y una contraseña.
- Además, un nuevo usuario puede registrarse en nuestra biblioteca proporcionando su nombre, su dni, nick y contraseña.
- En el sistema sólo puede haber un usuario conectado a la vez.

Ejercicio Diagrama de Clases (III)

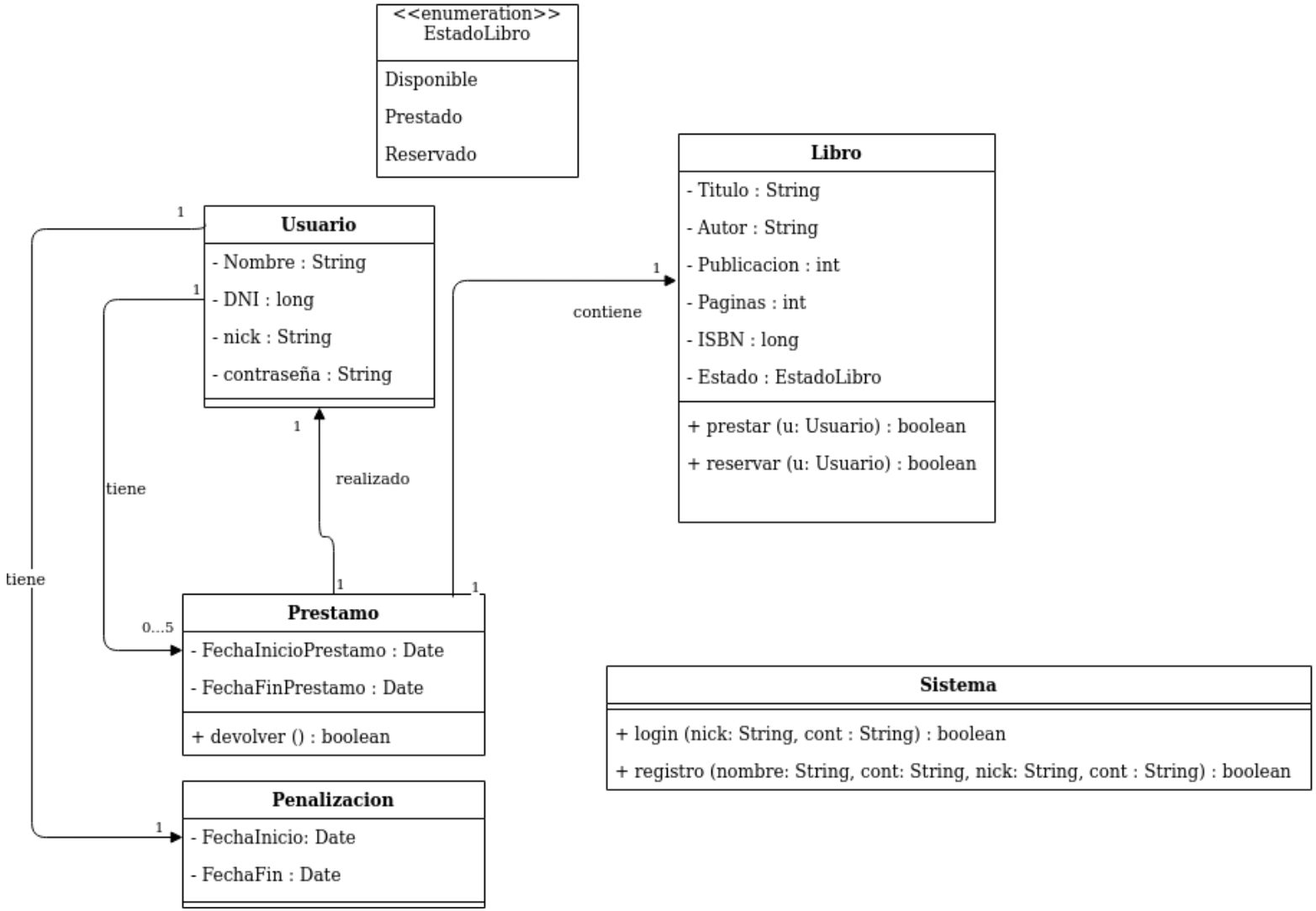


Ejercicio Diagrama de Clases (III)

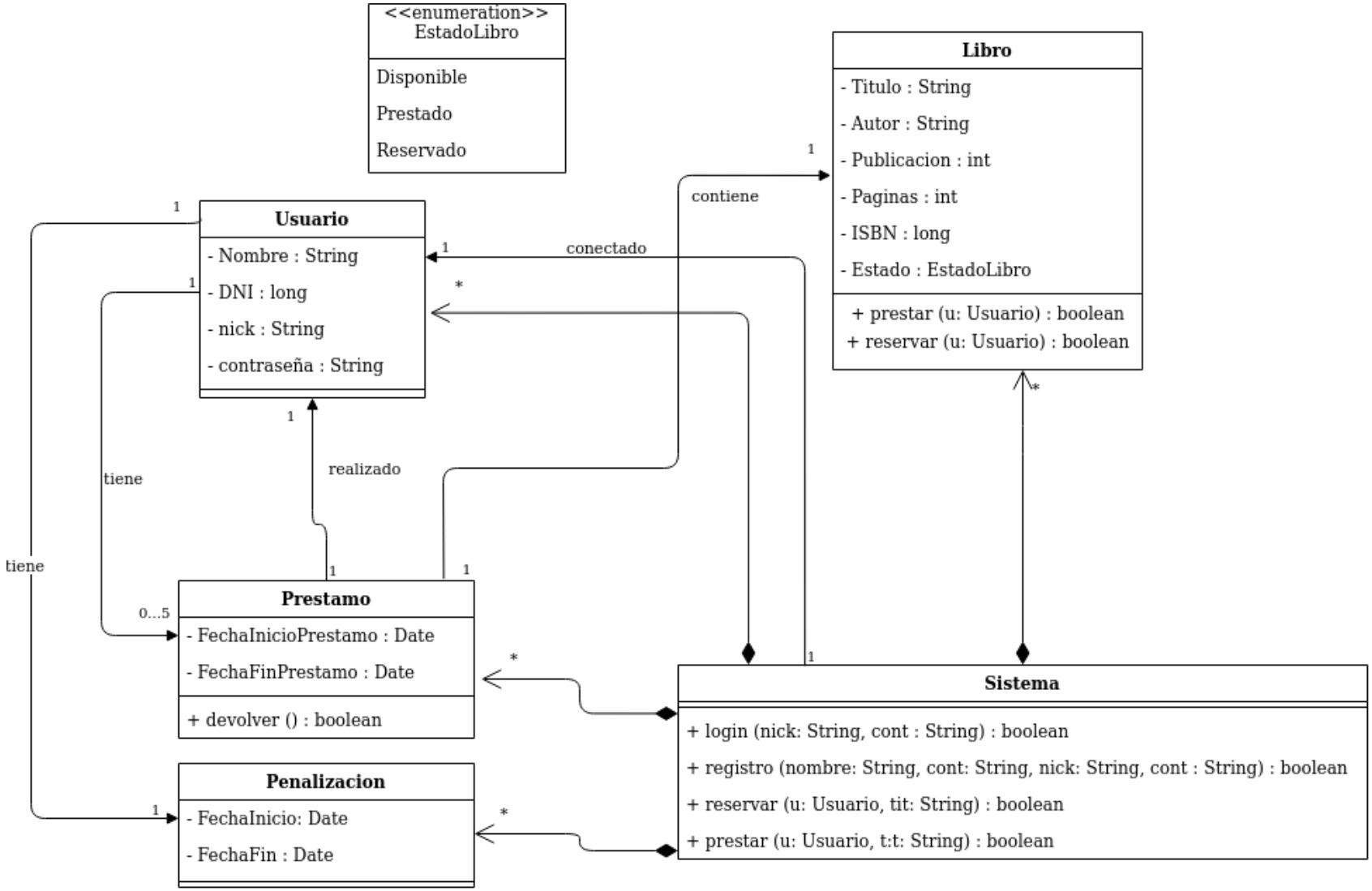


¡¡ Mal !!

Ejercicio Diagrama de Clases (III)



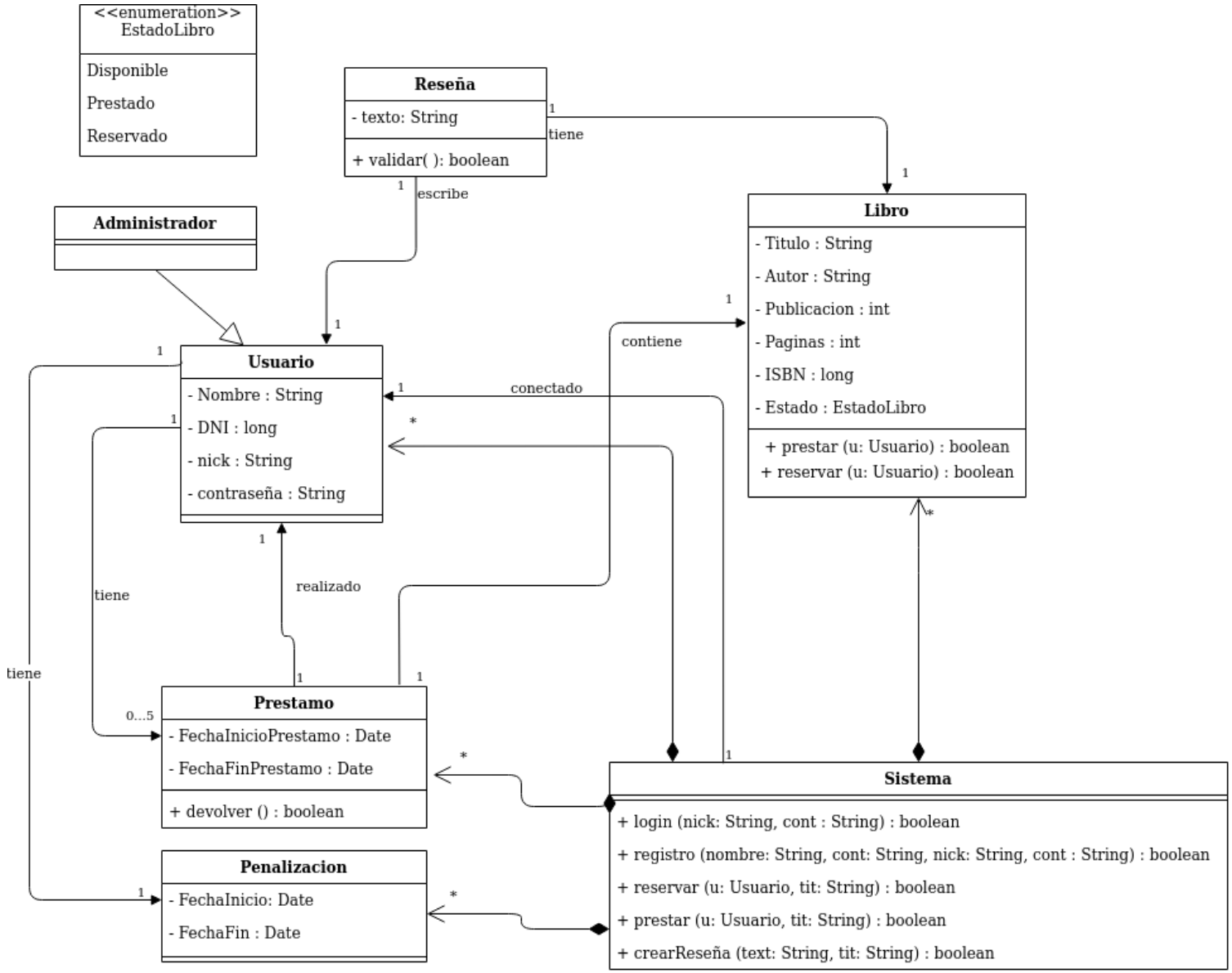
Ejercicio Diagrama de Clases (III)



Ejercicio Diagrama de Clases (IV)

- Los usuarios podrán escribir reseñas de los libros.
- Estas reseñas antes de que sean visibles para el resto de usuarios, deben ser validadas por un administrador.

Ejercicio Diagrama de Clases (IV)

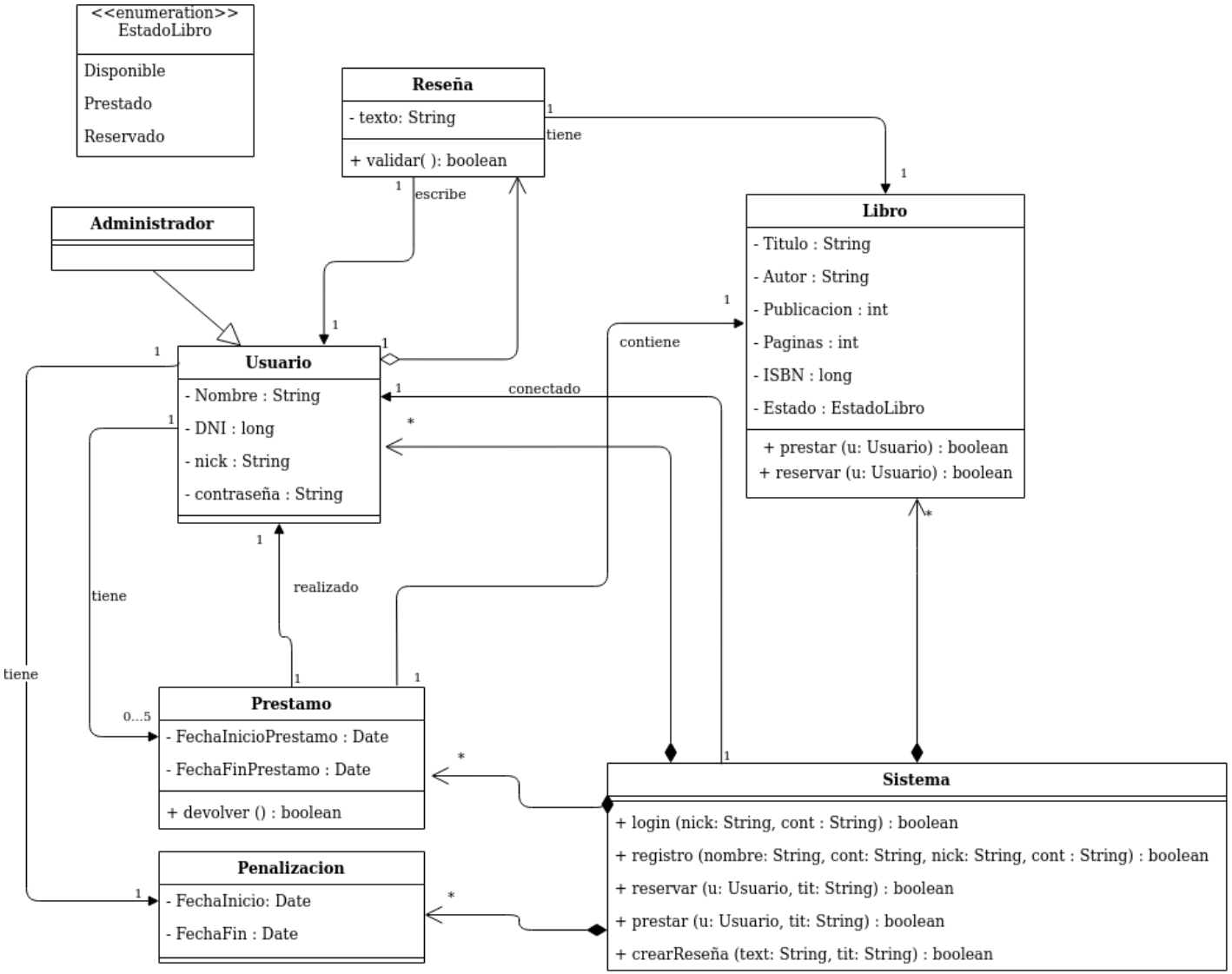


Ejercicio Diagrama de Clases (V)

- … el sistema tiene que almacenar todas las reseñas creadas…

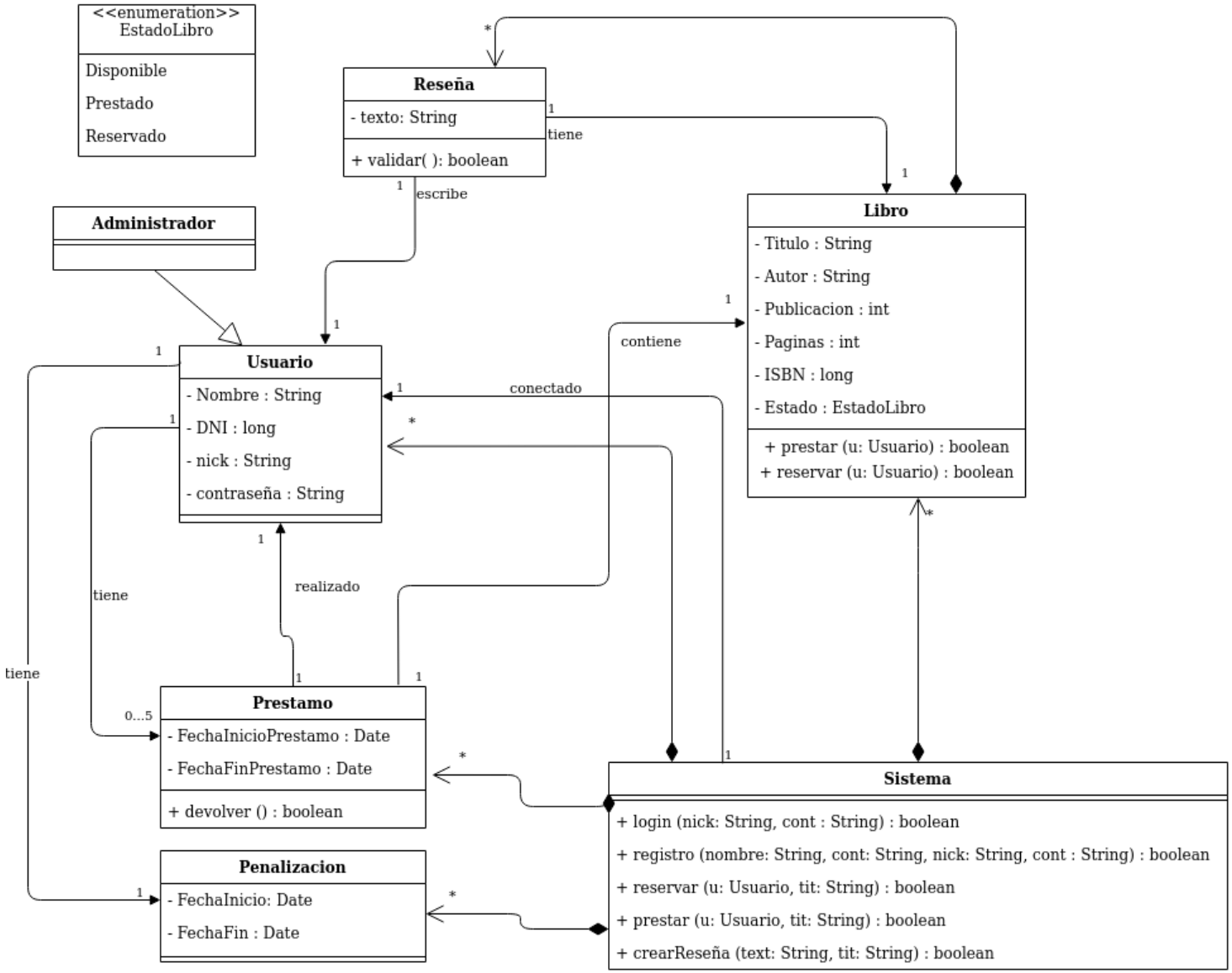
Ejercicio Diagrama de Clases (V)

- Opción a



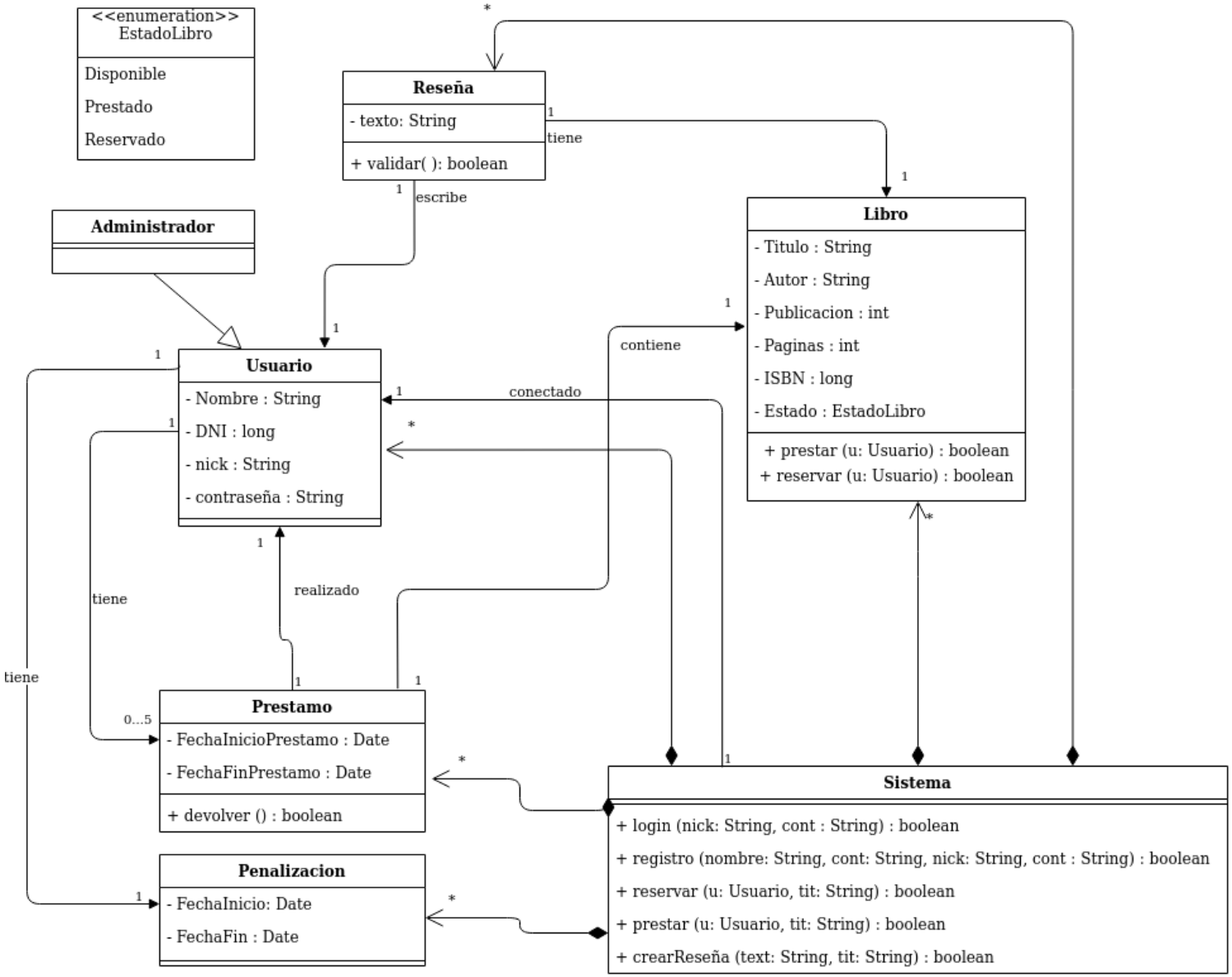
Ejercicio Diagrama de Clases (V)

- Opción b



Ejercicio Diagrama de Clases (V)

- Opción c



Ejercicio Diagrama de Clases

- Las tres soluciones presentadas son igual de válidas.
- En la práctica, nos decantaremos por una o por la otra dependiendo de razones prácticas.
- Por ejemplo, ¿a través de dónde vamos a acceder a los datos?

Ejercicio Diagrama de Clases

- Si sabemos que va a ser frecuente acceder a todas las reseñas de un libro…
- … con la opción a (los usuarios contienen las reseñas) tendríamos que recorrer todos los usuarios y por cada usuario recorrer todas sus reseñas para recuperar aquellas que se corresponden con el libro.
- … con la opción b (los libros tienen las reseñas) sólo tendríamos que acceder al libro en cuestión para tener las reseñas.
- … con la opción c (el sistema tiene todas las reseñas sueltas) tendríamos que recorrer todas las reseñas y ver si se corresponden al libro que nos interesa.

Diseño UML

- Diferentes tipos de diagramas:
  - Modelado de la estructura:
    - Diagrama de clases.
  - **Modelado de comportamiento**:
    - Diagrama de Transición de Estados.
    - Diagrama de Secuencia.

- Matriz de Trazabilidad

Modelado de Comportamiento

- Un diagrama de clases nos dice la estructura de la aplicación, pero no el comportamiento:
  - ¿qué hace cada método?
  - ¿cómo cambia el estado de un objeto ante invocaciones de métodos?
  - ¿cómo colabora un conjunto de objetos para realizar una tarea?

Modelado de Comportamiento

- Un diagrama de clases nos dice la estructura de la aplicación, pero no el comportamiento:
  - ¿qué hace cada método?
    - Pseudocódigo
    - Diagrama de actividad.
  - ¿cómo cambia el estado de un objeto ante invocaciones de métodos?
    - Diagrama de transición de estados (statecharts)
  - ¿cómo colabora un conjunto de objetos para realizar una tarea?
    - Diagramas de secuencia

Diagrama de Transición de Estados

- Van asociados a una clase.
- Describen cómo evoluciona cuando se le invocan métodos.
- Similar a un autómata finito.

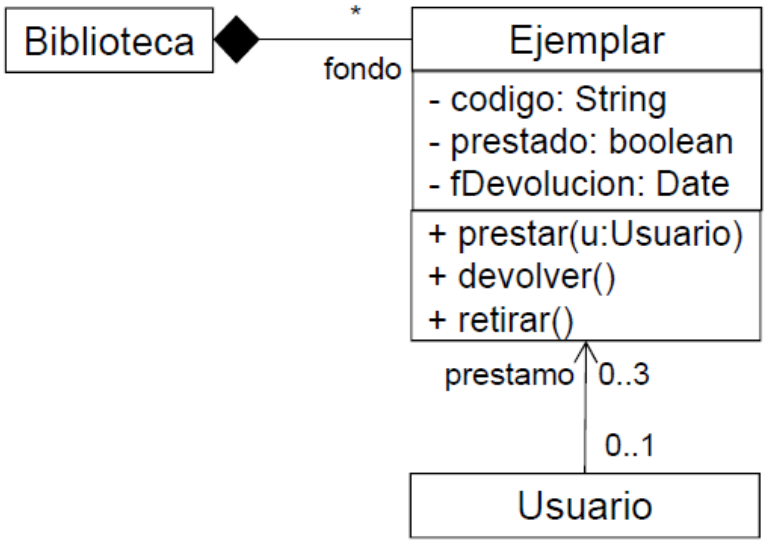


Diagrama de clases

Diagrama de Transición de Estados

- Van asociados a una clase.
- Describen cómo evoluciona cuando se le invocan métodos.
- Similar a un autómata finito.

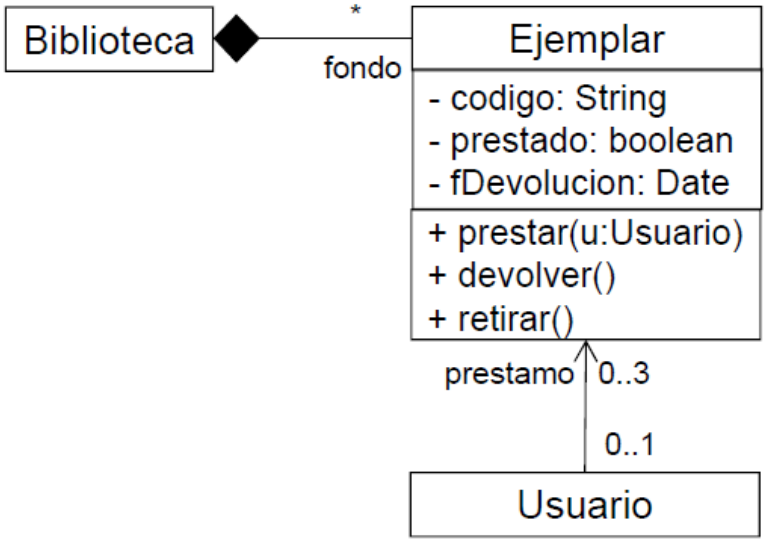


Diagrama de clases

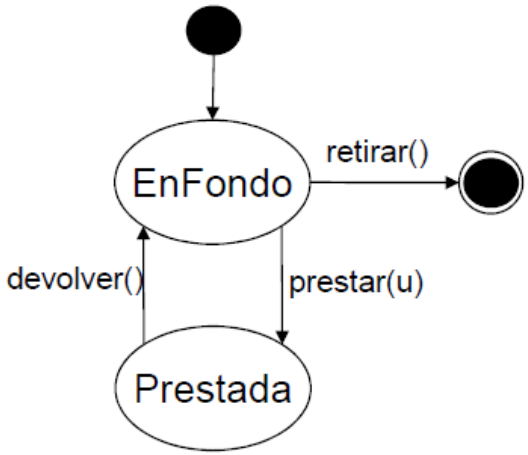


Diagrama de transición de estados (clase Ejemplar)

Diagrama de Transición de Estados

- Estados jerárquicos.
- Guardas y acciones en las transiciones.

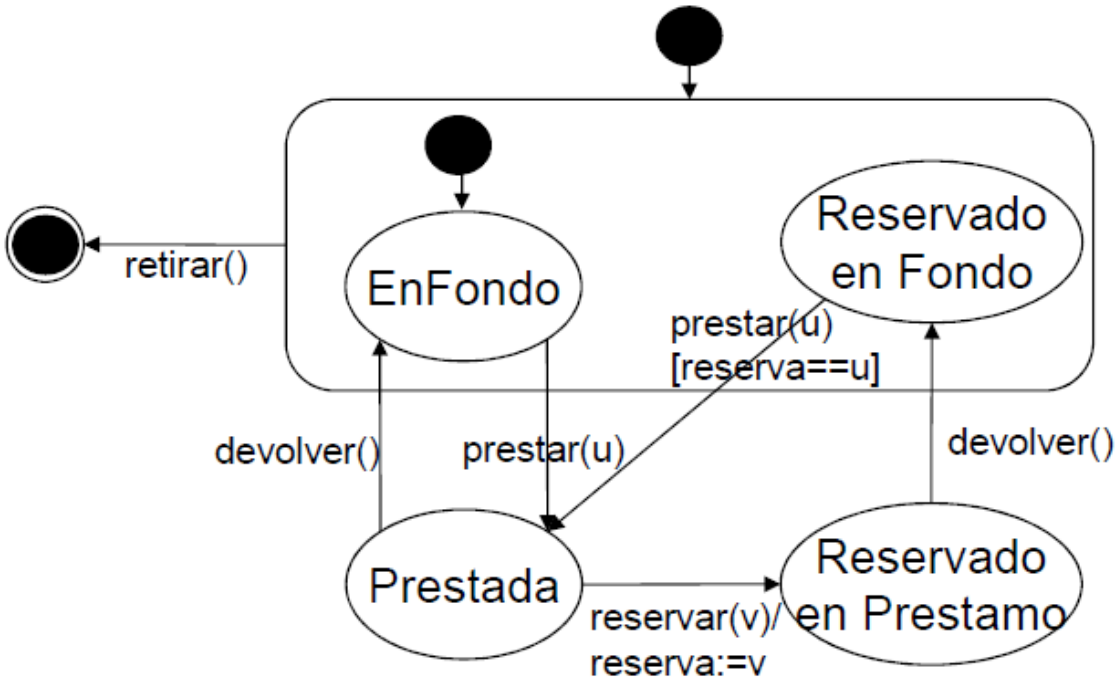


Diagrama de Secuencia

- Representa una secuencia de mensajes que intercambia **un conjunto de objetos.**
- Cada objeto tiene una línea de vida (representada verticalmente)
  - El paso del tiempo se representa de manera descendente.
  - Invocación de mensajes: flechas entre líneas de vida.
  - Cajas de activación.

Diagrama de Secuencia

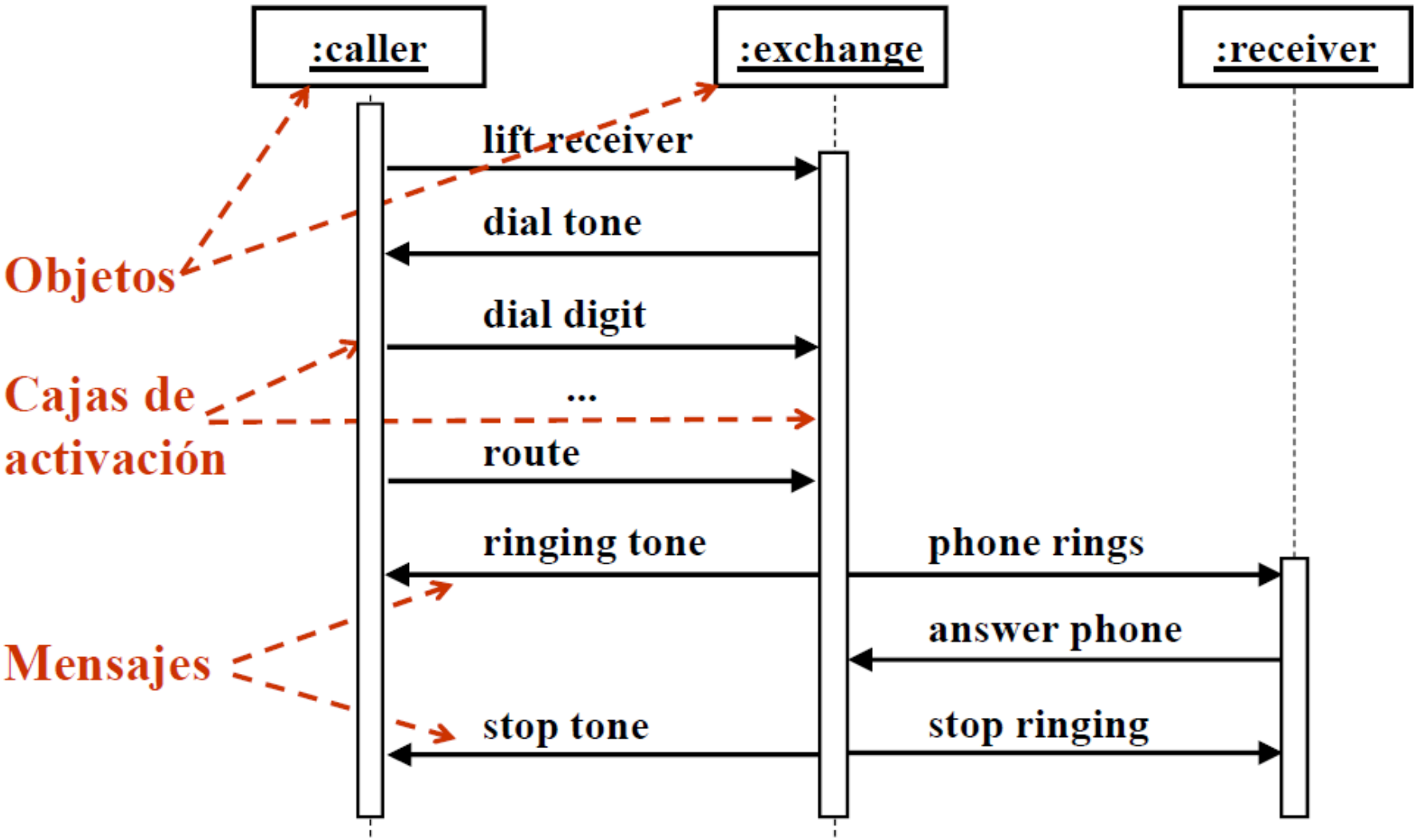


Diagrama de Secuencia

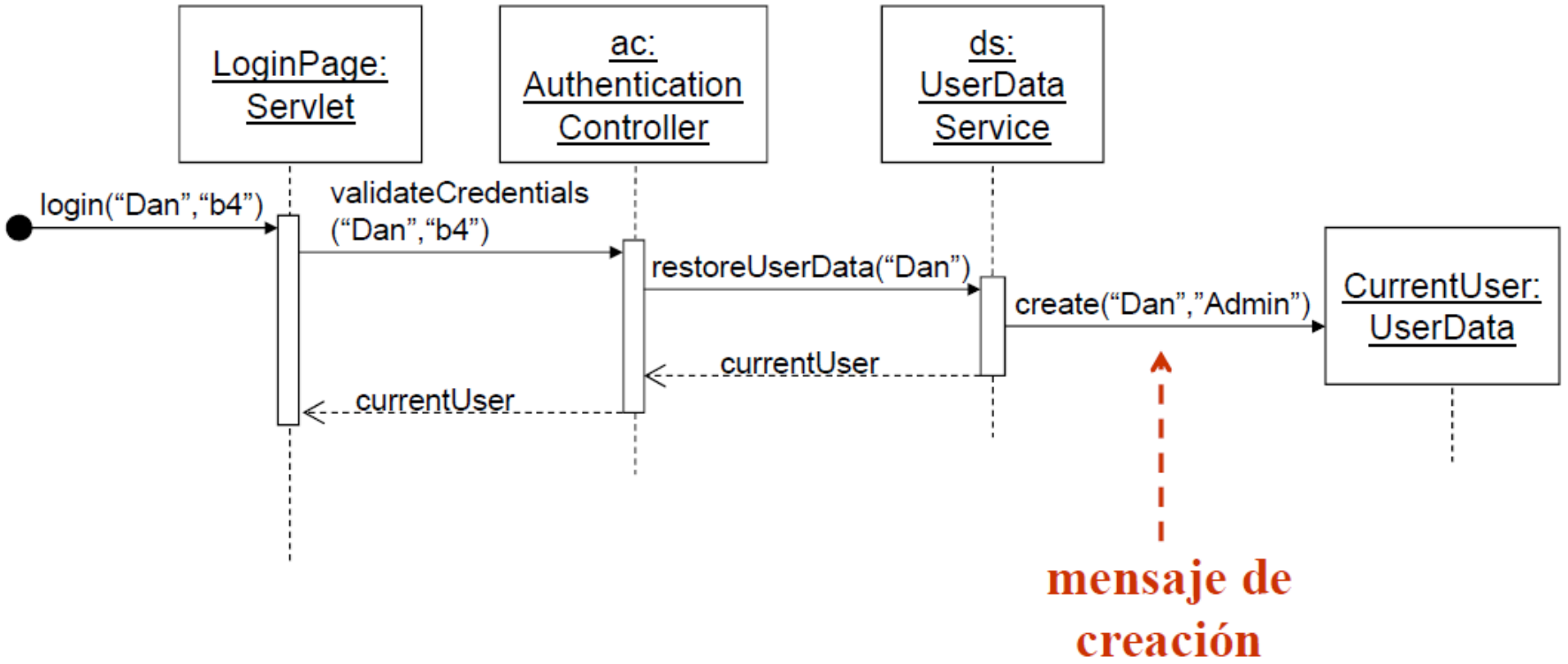
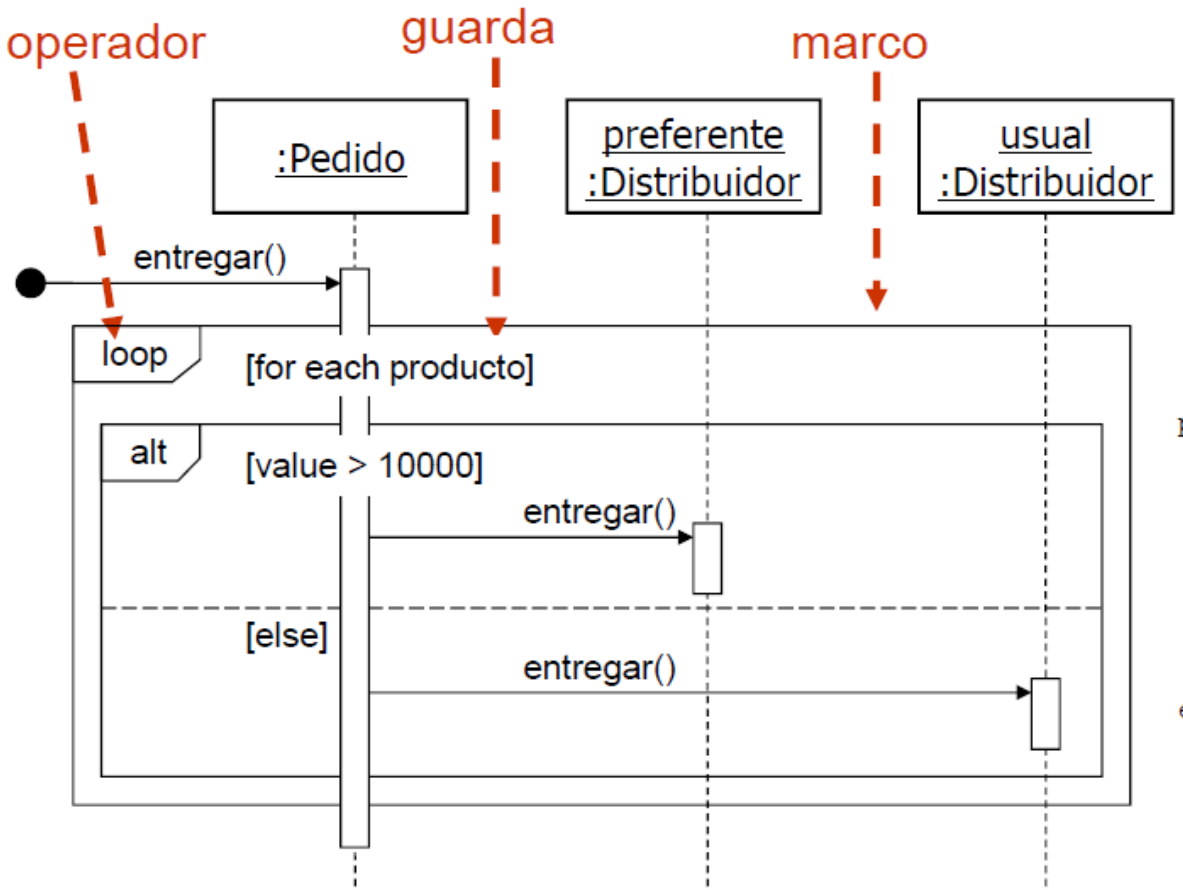


Diagrama de Secuencia (operadores)




```
procedure entregar()
   foreach producto:
      if producto.value>10000
         preferente.entregar()
      else
         usual.entregar()
      end if
   end for
end procedure
```

Diagrama de Secuencia (operadores)

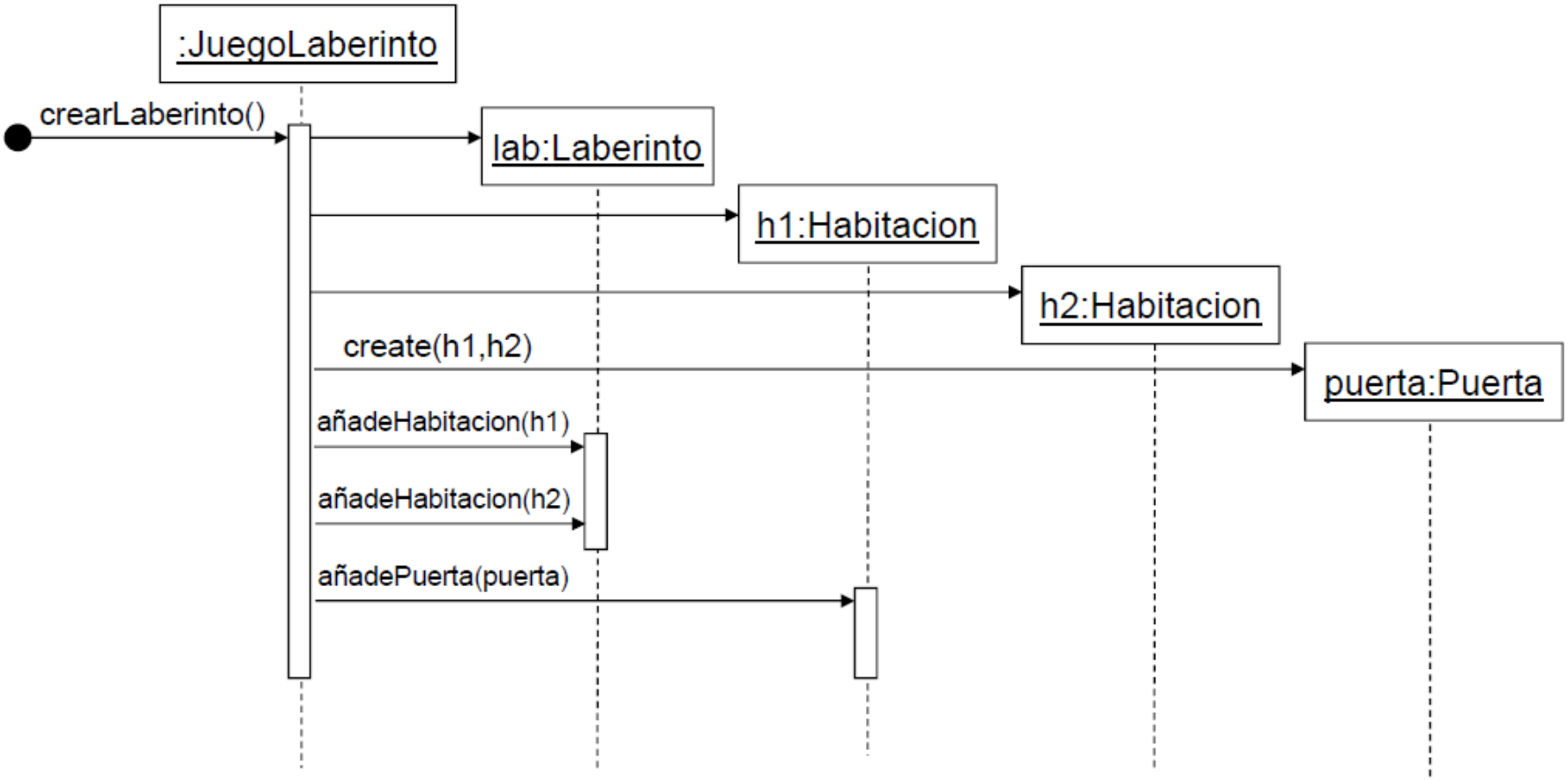
- **Alternativa (alt)**: elección (mediante una guarda) de una interacción. Múltiples fragmentos, sólo se ejecuta el que cumple la guarda.
- **Opción (opt)**: equivale a un operador *alt* con un solo fragmento. Equivalente al if en programación.
- **Bucle (loop)**: el fragmento se ejecuta múltiples veces. La guarda indica cómo realizar la iteración.
- Existen otros como *neg, par, critical, ref..*

Diagrama de Secuencia (Ejercicio)

- **Realizar el diagrama de secuencia del siguiente código:**

```
public class JuegoLaberinto {
   public Laberinto crearLaberinto () {
      Laberinto lab = new Laberinto();
      Habitacion h1 = new Habitacion();
      Habitacion h2 = new Habitacion();
      Puerta puerta = new Puerta(h1, h2);
      lab.añadeHabitacion(h1);
      lab.añadeHabitacion(h2);
      h1.añadePuerta(puerta);
      return lab;
   }
}
```

Diagrama de Secuencia (Ejercicio)



Diseño UML

- Diferentes tipos de diagramas:
  - Modelado de la estructura:
    - Diagrama de clases.
  - Modelado de comportamiento:
    - Diagrama de Transición de Estados.
    - Diagrama de Secuencia.

- **Matriz de Trazabilidad**

Matriz de Trazabilidad

- Al finalizar el diseño (detallado) de la aplicación, es necesario comprobar que todos los requisitos educidos en la fase de análisis del problema tienen su correspondiente representación en el dominio de la solución a implementar.
- Matriz de Trazabilidad de Requisitos

	Elementos Funcionales			
Requisitos	Clase1.Método1	Clase1.Método2	…	ClaseN.MétodoM
Requisito 1	X	X		
Requisito 1.1		X		
…				
Requisito N	X			X
Requisito N.M				X