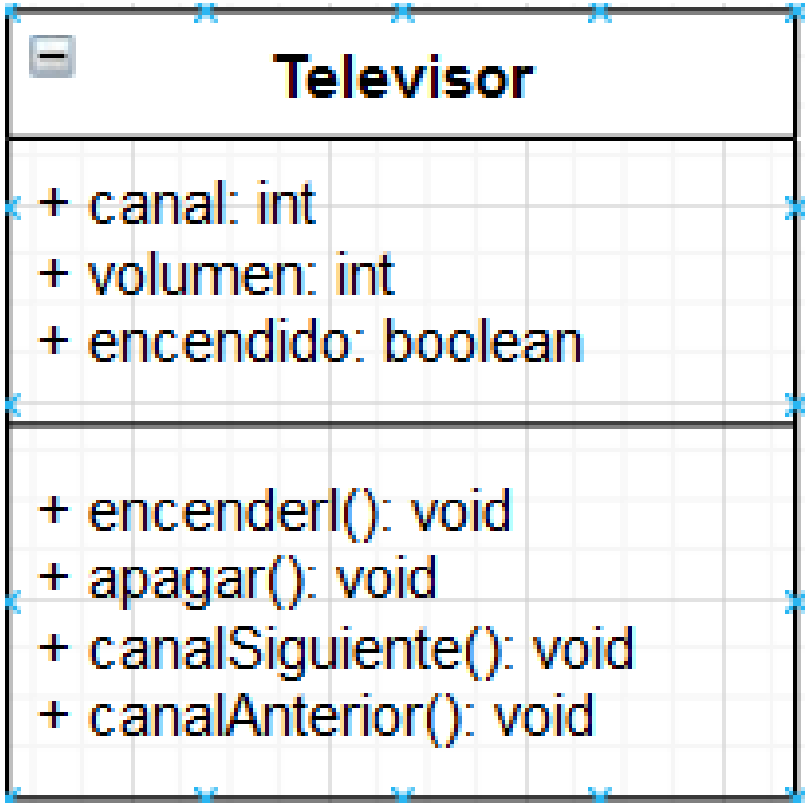
Unidad N° 1: Introducción a la Programación Orientada a Objetos

Conceptos básicos

- Clase
- Atributo
- Método
- Objeto
- Estado
- Mensaje

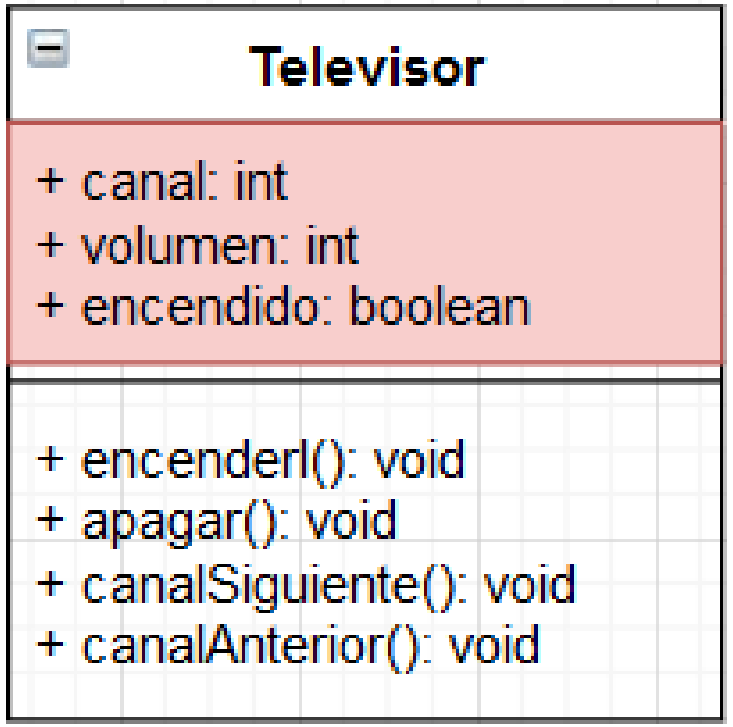
Clase

- Representación de un objeto o entidad de la vida real, compuesta por las características específicas de dicho objeto/entidad (*atributos*) y por las operaciones que definen el comportamiento de la clase (*métodos*).
- Generalización o abstracción de un tipo de objetos en específico.



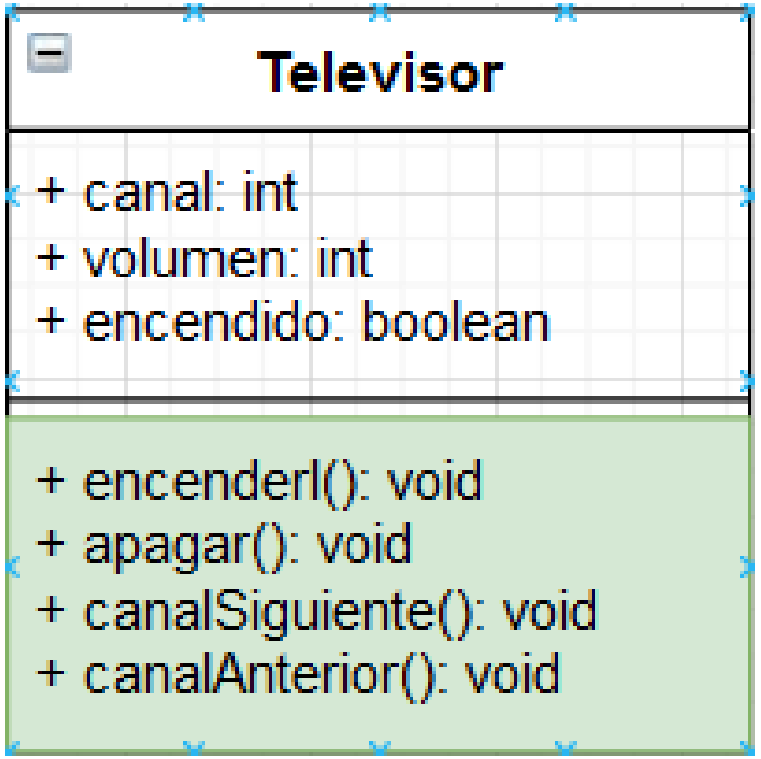
Atributo

- Un atributo es una característica individual que diferencia a un objeto de otro y determinan la estructura de la clase.
- Dentro de la clase, los atributos se definen como variables individuales que poseen un tipo de dato para cada una de ellas.



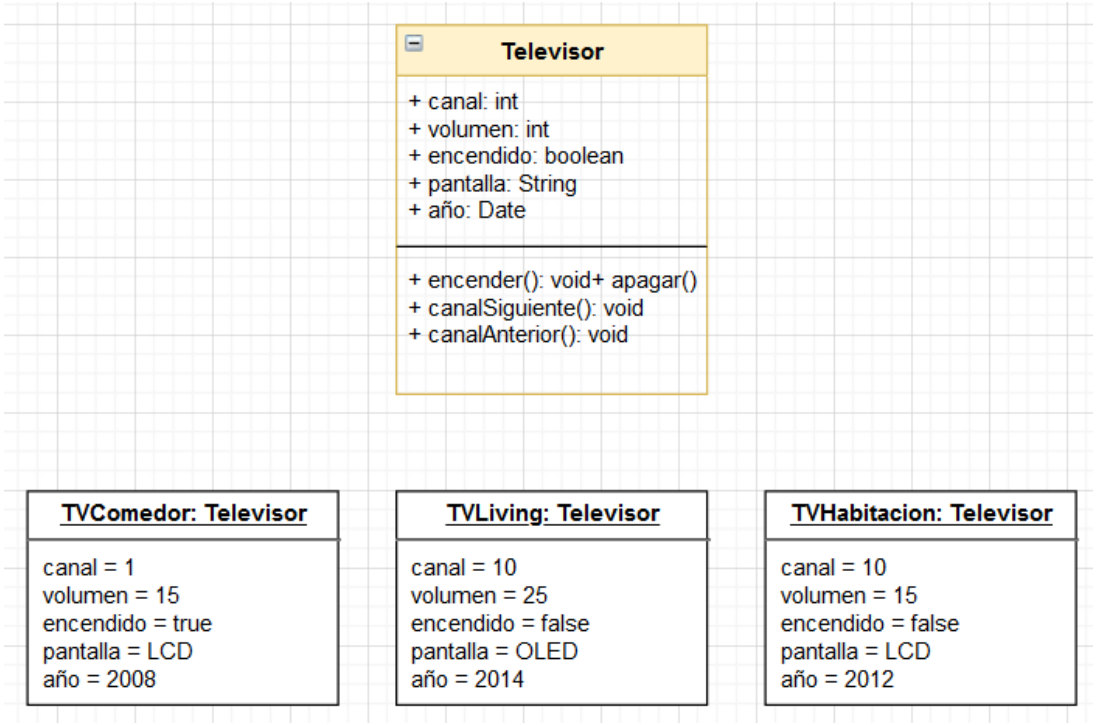
Método

- Un método es un conjunto de instrucciones que realizan una determinada tarea, y por lo general se ejecuta tras la recepción de un mensaje.
- Dentro de la clase, los métodos se definen como funciones que poseen un tipo de dato de retorno y una visibilidad explícita.

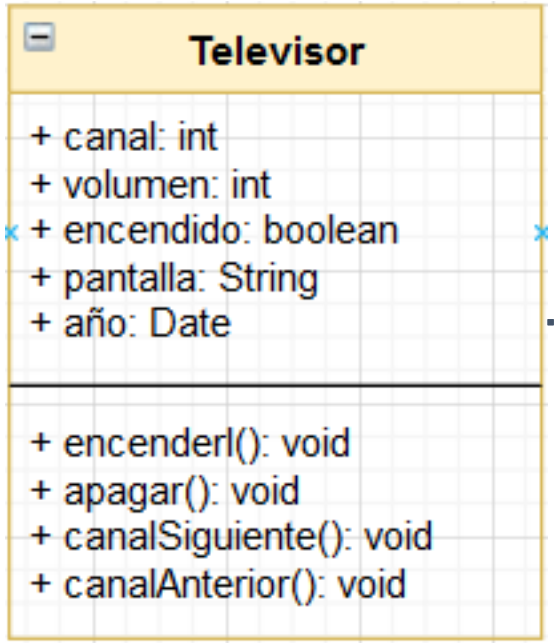


Objeto

- Es una instancia de la clase; es una entidad creada a partir de una clase y posee su propio conjunto de datos y su conjunto de operaciones.
- Al proceso de creación de un objeto se lo denomina ***instanciación***.

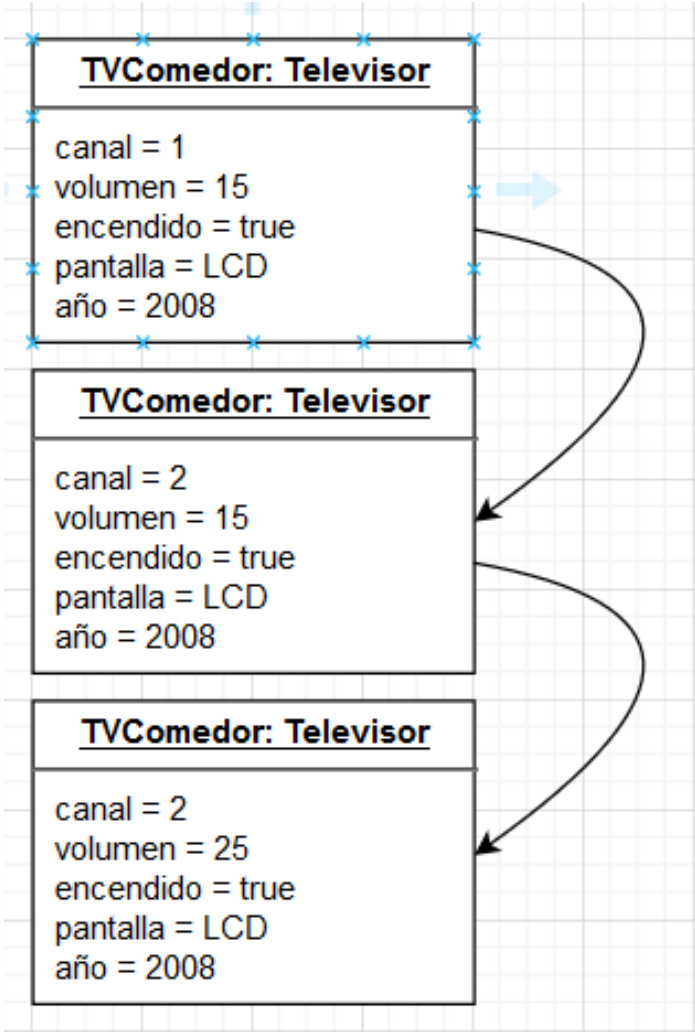


Ejemplo de objetos de una misma clase



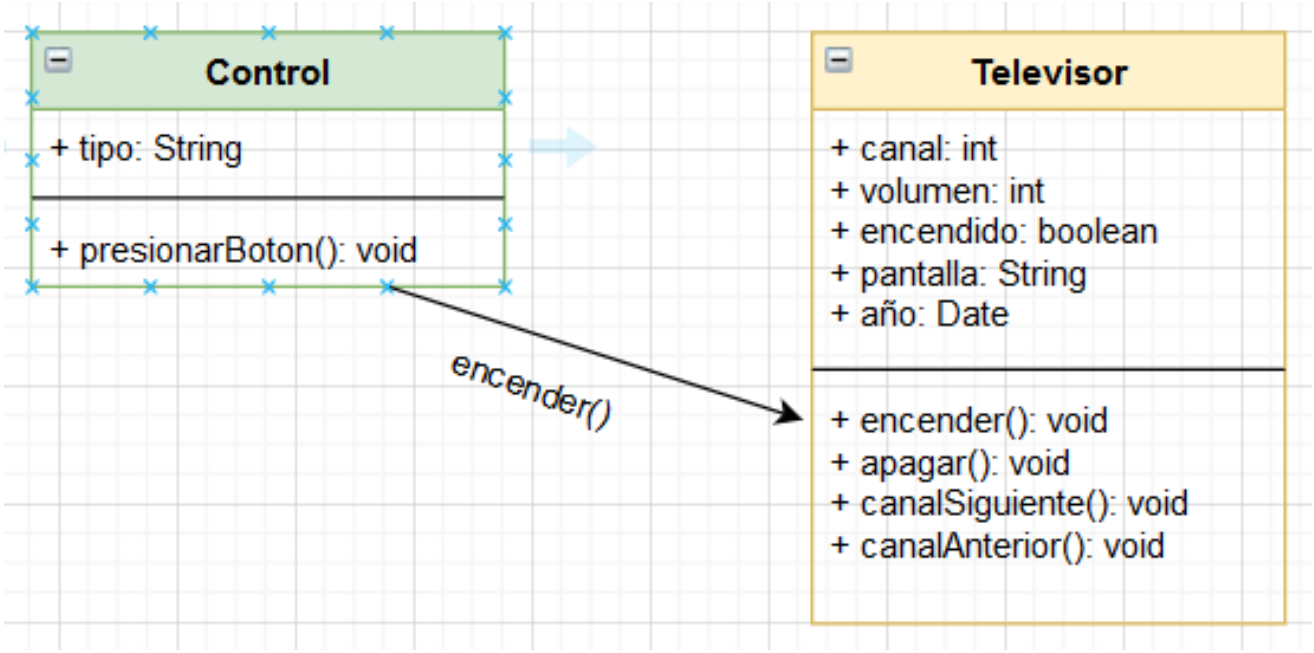
Estado

- Es el conjunto de valores de los atributos de un objeto en un determinado tiempo.
- Cuando cambia el valor de un atributo, cambia el estado de dicho objeto.



Mensaje

- Es una orden dirigida desde un objeto hacia un segundo objeto para invocar la ejecución de un método de este último.
- Esto se conoce como «enviar un mensaje».



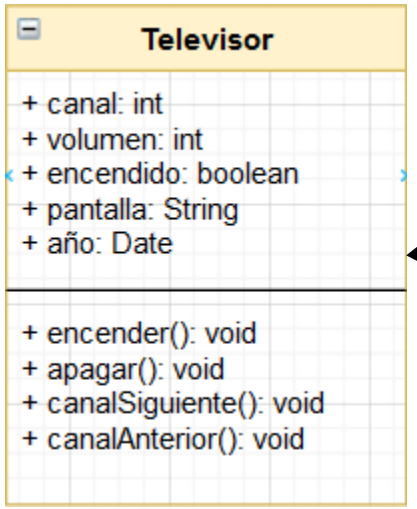
Métodos especiales

- **Constructor**: se ejecuta automáticamente al crear un objeto; inicializa los valores de los atributos. Se define con el mismo nombre de la clase.
- **Getter**: permite acceder al valor de un atributo; se define uno por cada atributo necesario empezando con la palabra "*get*" + el nombre del atributo (en el caso de los booleanos, empieza por "*is*").
- **Setter**: permite modificar el valor de un atributo; se define uno por cada atributo necesario empezando con la palabra "*set*" + el nombre del atributo.
- **Destructor**: se ejecuta automáticamente cuando un objeto es eliminado; permite realizar tareas de mantenimiento. No es tan usual (en Java no es necesario implementarlo).

Características fundamentales

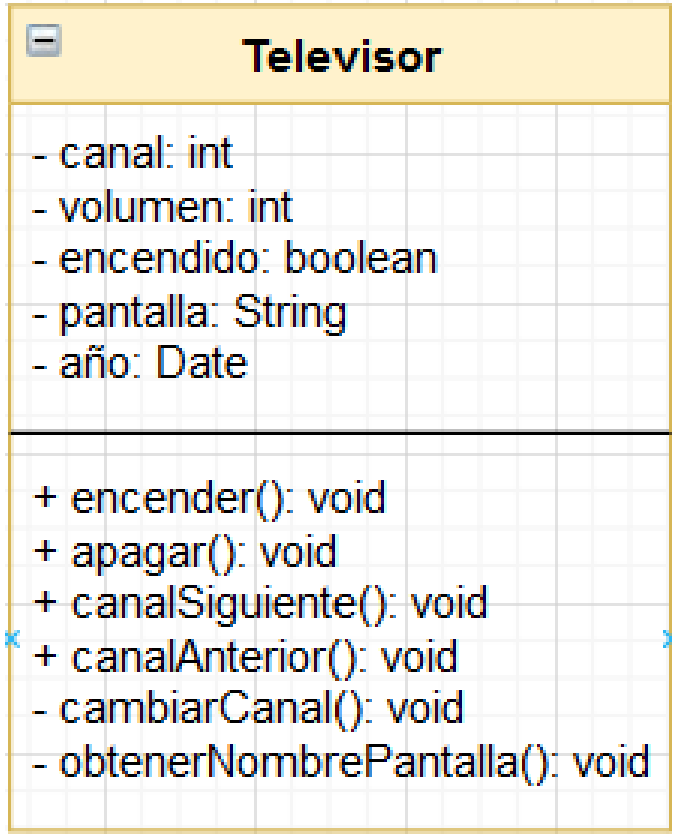
Abstracción

- Proceso que permite seleccionar las características relevantes de una clase según el sistema en el cual se verá representada.
- Las clases y objetos de un sistema son entidades abstractas que puede comunicarse con otras entidades sin revelar su funcionamiento interno.
- El objetivo es revelar solamente la información que es relevante y ocultar la complejidad



Encapsulamiento

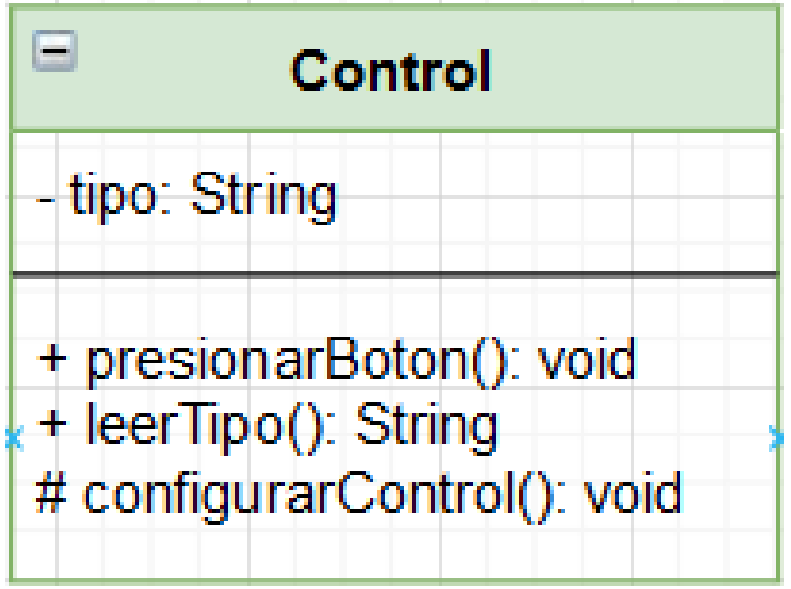
- Capacidad de agrupar datos y operaciones relacionadas bajo la misma unidad de programación (clase) y limitar el acceso de clases externas a la implementación de su estructura; esto último se realiza mediante los modificadores de visibilidad.
- El encapsulamiento define qué mensajes pueden ser enviados a una clase en particular; permite ocultar los detalles internos de un objeto y controlar quién puede acceder a su información.



Modificadores de visibilidad

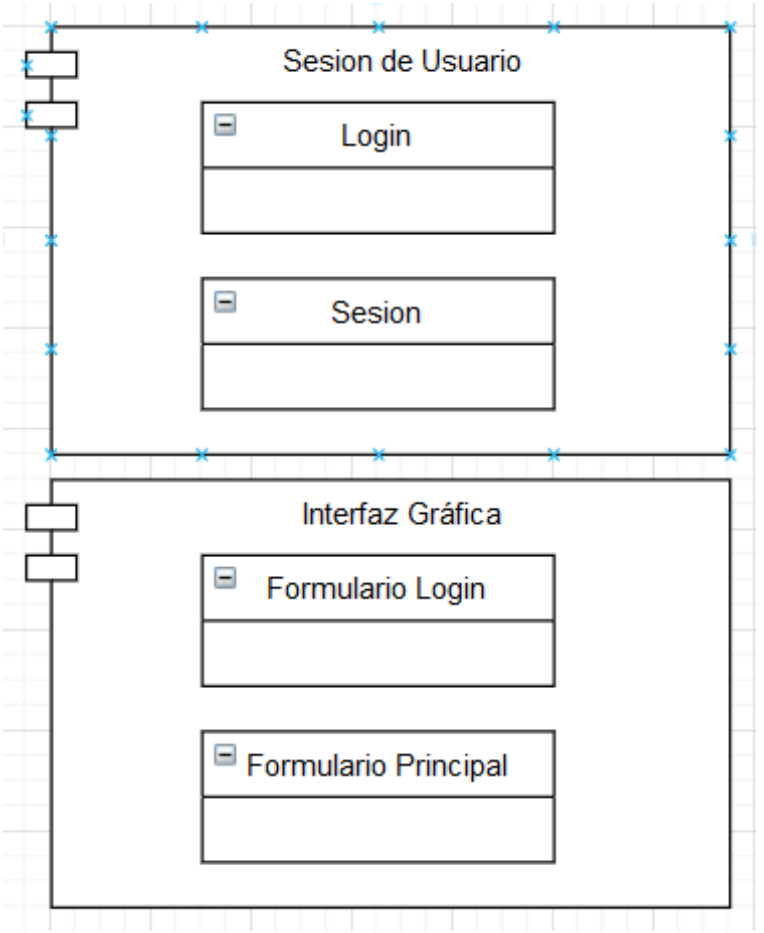
Los modificadores de visibilidad especifican las reglas de acceso a un atributo o un método de clase.

- **public**: define que el elemento puede ser accedido desde cualquier clase.
- **private**: define que el elemento puede ser accedido solamente desde la clase que lo define.
- **protected**: define que el elemento puede ser accedido desde la clase que la define o sus sub-clases.



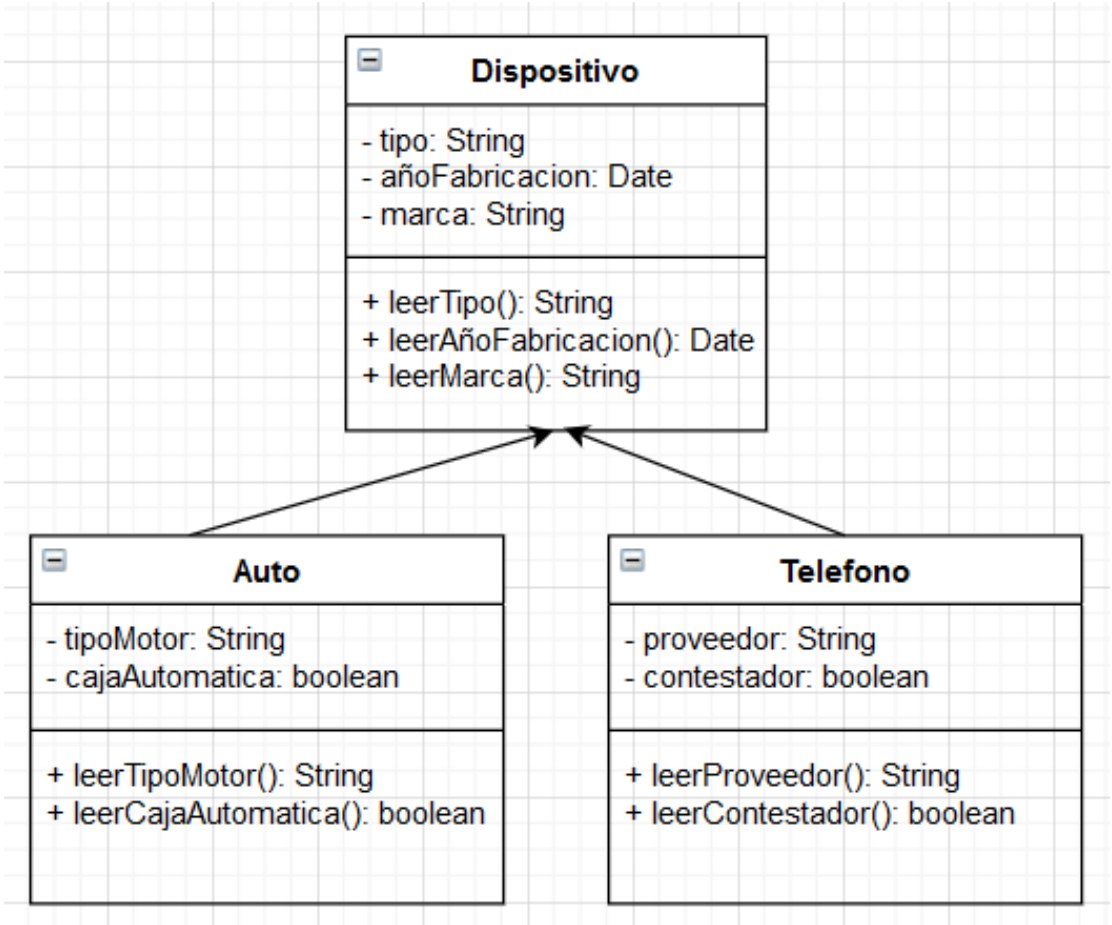
Modularidad

- Propiedad que permite dividir una aplicación o sistema en partes más sencillas llamadas módulos, los cuales permiten agrupar clases que posean un mismo propósito.
- Los módulos permiten reducir la complejidad del código y definir responsabilidades únicas (*Principio de Responsabilidad Simple*).



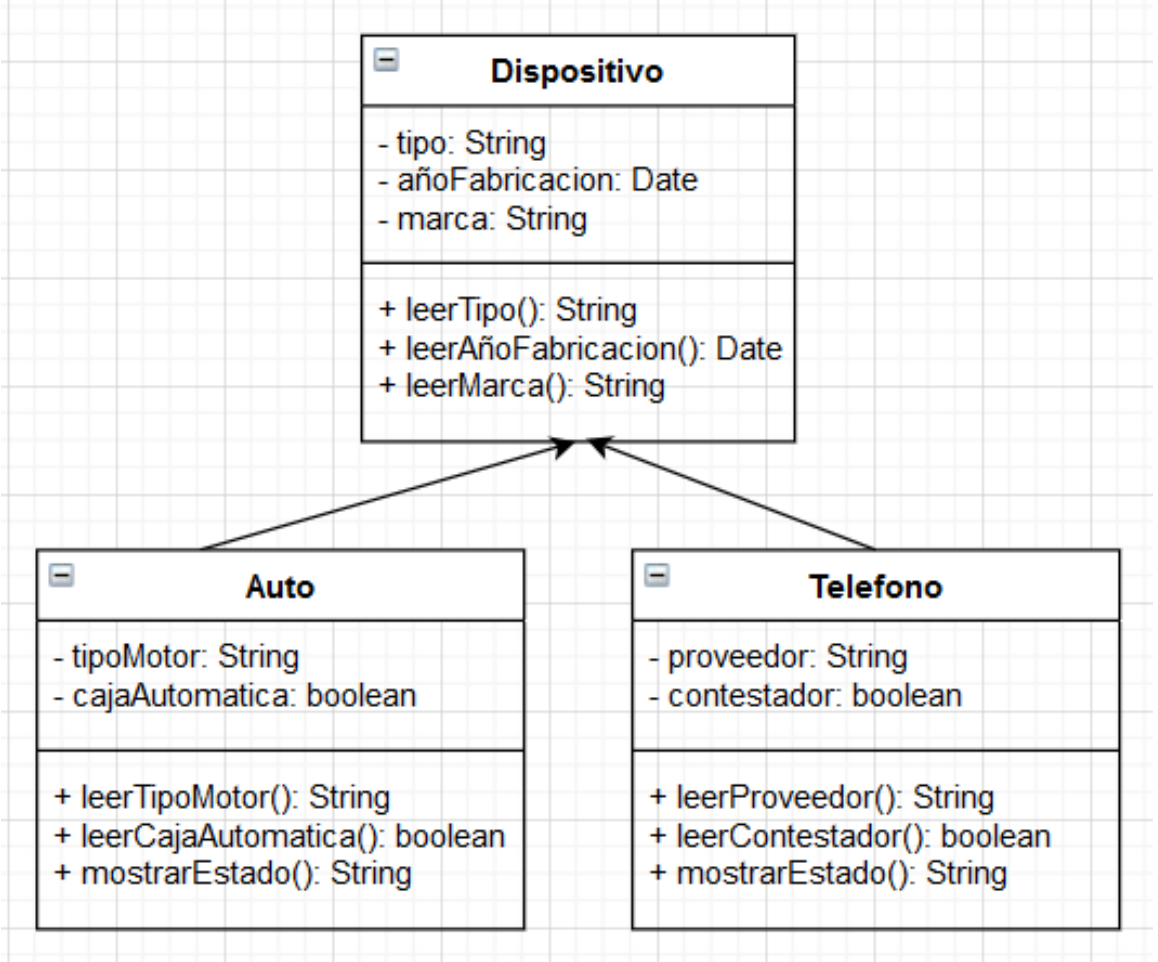
Herencia

- Es un mecanismo que permite definir clases en base a otras ya existentes, de modo que la subclase obtiene los atributos y métodos de la superclase. De esta forma se va formando un *árbol de jerarquía*.
- ***Subclase → Clase hija.***
- ***Superclase → Clase padre.***



Polimorfismo

- Mecanismo que permite que un método responda de distintas maneras a un mensaje. Esto permite que diversas clases definan el mismo método pero que este opere de manera distinta según su contexto.
- ***«Poli»*** ***→ Muchos.***
- ***«Morfos»*** ***→ Forma.***



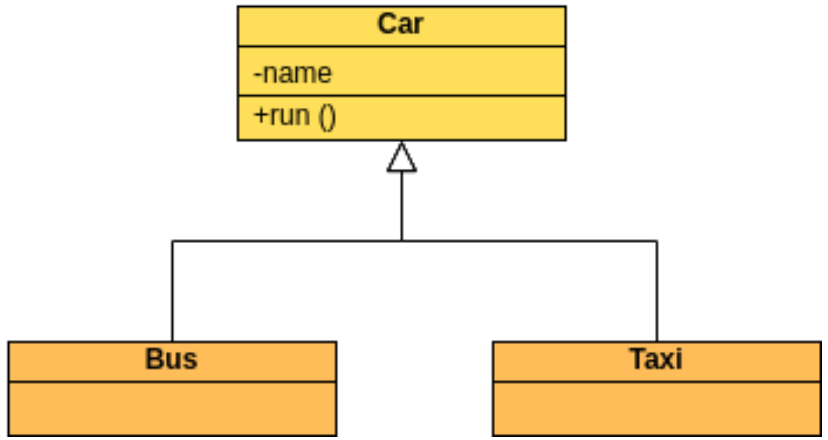
Relaciones entre clases

Tipos de relaciones

- En la POO las clases no existen de forma aislada, sino que interactúan con otras clases con el fin de lograr el objetivo del sistema.
- Los tipos de relaciones disponibles son:
  - Generalización (Herencia)
  - Realización (Implementación)
  - Agregación
  - Composición

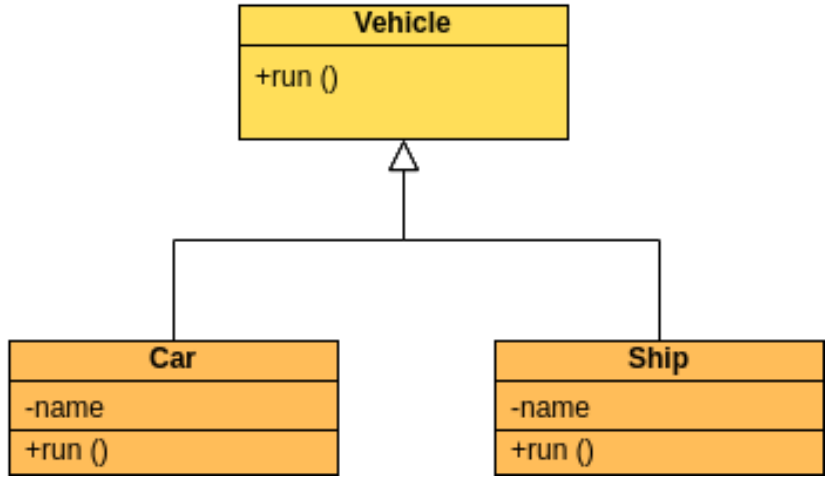
Herencia/Generalización

- Describe la relación entre una clase padre y una clase hija.
- La subclase hereda todas las funciones de la clase principal y la clase principal tiene todos los atributos, métodos y subclases. Las subclases contienen información adicional además de la misma información que la clase principal.
- Se indica la relación con una línea continua con punta de flecha triangular.



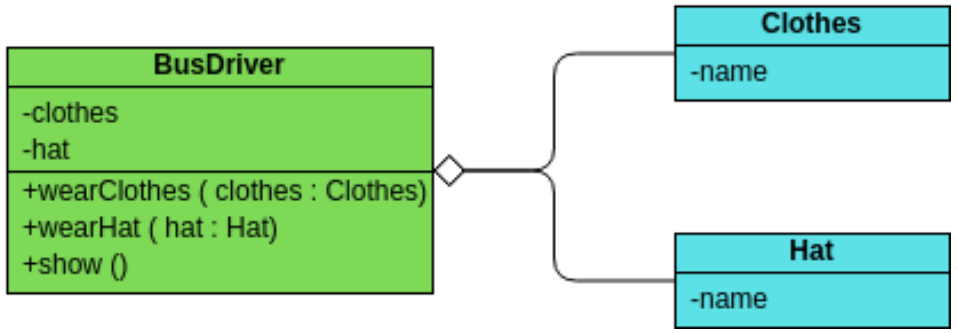
Realización/Implementación

- Describe la relación entre una interface y las clases de implementación.
- **Una interfaz** (incluida una **clase abstracta**) es una colección de métodos. En una relación de implementación, una clase implementa una interfaz y los métodos de la clase implementan todos los métodos de la declaración de la interfaz.
- Se indica la relación con una línea continua con punta de flecha triangular (igual que la herencia)



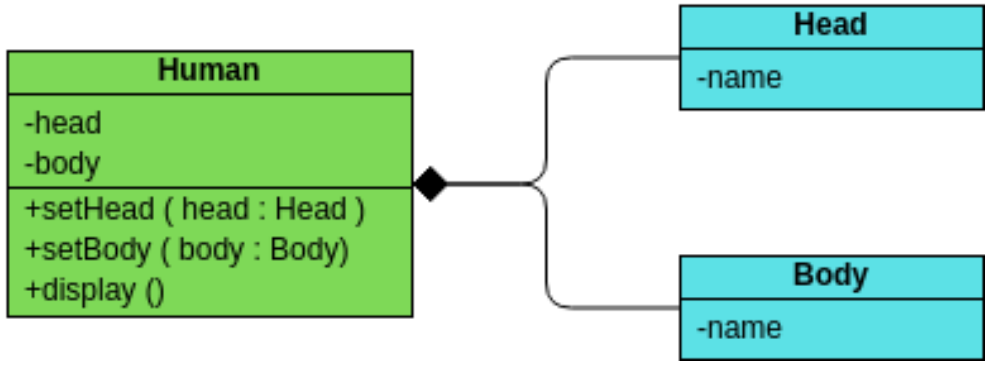
Agregación

- Relación de tipo "todo/parte", en la cual una clase contiene a otra, pero las instancias de las clases involucradas **pueden existir de manera independiente**.
- La clase que contiene es conocida como la clase "todo" y la clase contenida como la clase "parte".
- En la agregación, la vida de las instancias contenidas no depende de la existencia de la clase contenedora.
- Se indica la relación con una línea continua con una punta de rombo blanco



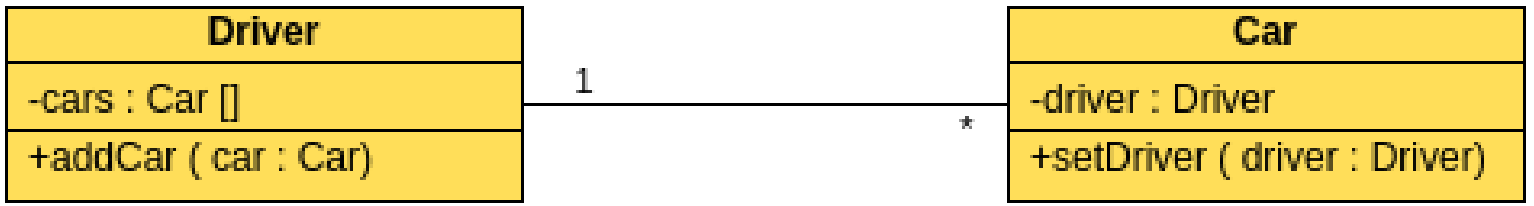
Composición

- **La relación entre el todo y la parte, pero el todo y la parte no se pueden separar**.
- La existencia de la clase "parte" está estrictamente vinculada a la clase "todo". Si la clase "todo" es destruida, la clase "parte" también lo será, ya que la clase "parte" no puede existir independientemente de la clase "todo“
- Se indica la relación con una línea continua con una punta de rombo negro



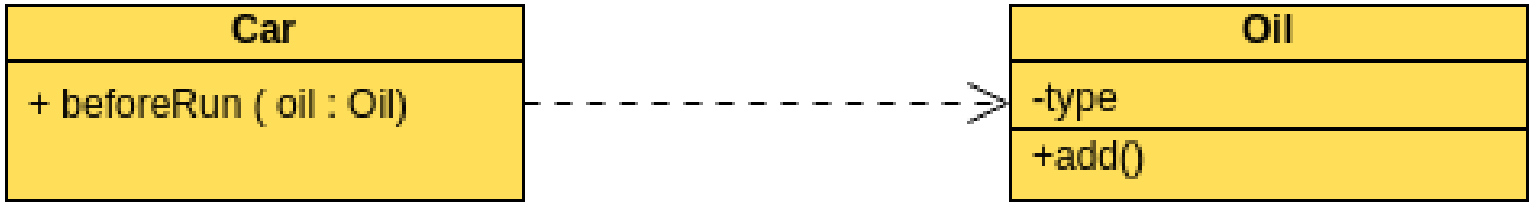
Asociación

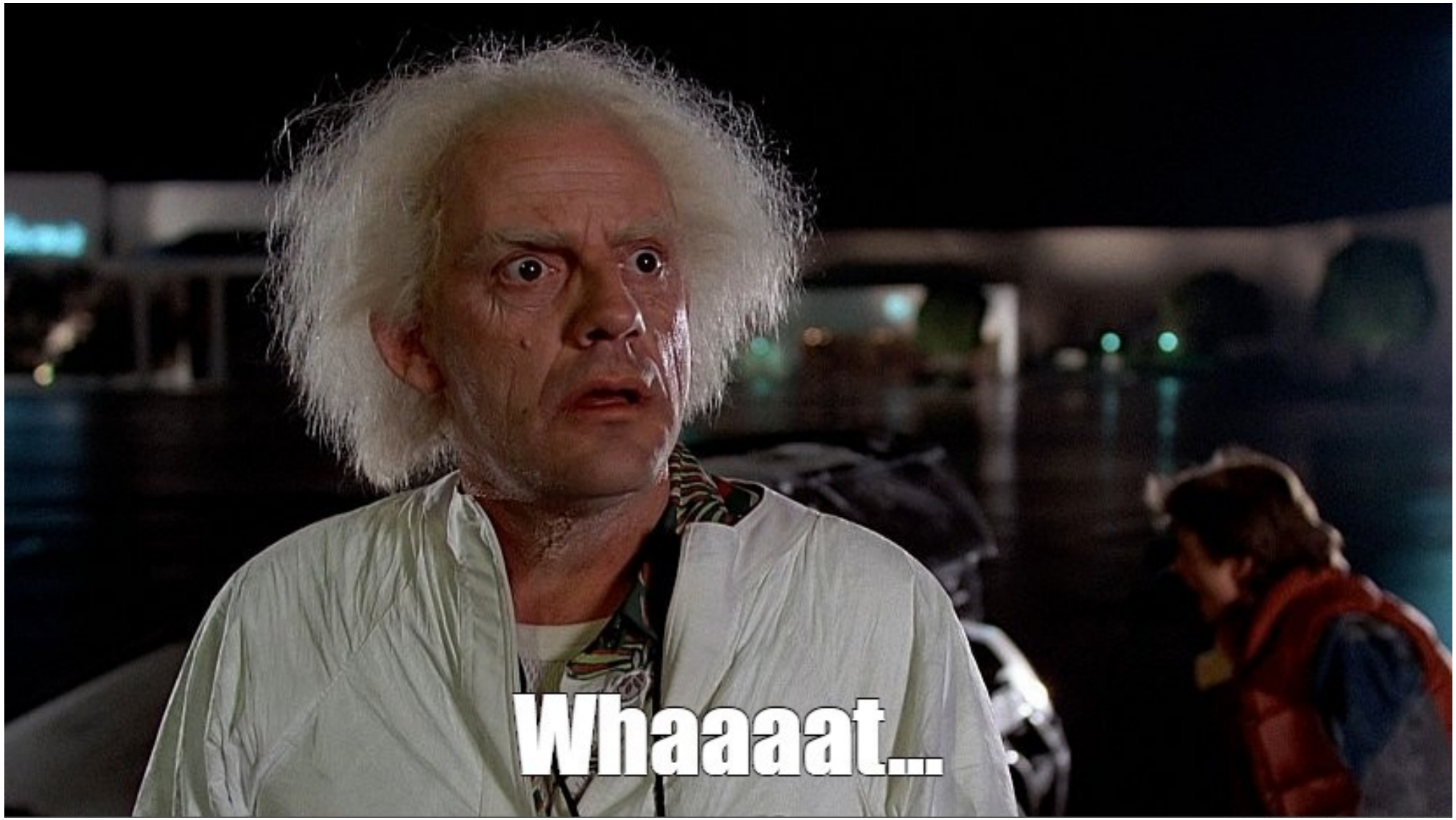
- Indica que **una propiedad de una clase contiene una referencia a una instancia (o instancias) de otra clase**.
- La asociación es la relación **más utilizada** entre una clase y otra clase, lo que significa que existe una conexión entre un tipo de objeto y otro tipo de objeto. **Las combinaciones y agregaciones también pertenecen a las relaciones asociativas** , pero las relaciones entre clases de afiliaciones son más débiles que las otras dos.
- Se indica la relación con una línea continua (con cardinalidad opcional).



Dependencias

- En la mayoría de los casos, **las dependencias se reflejan en los métodos de una clase que utilizan el objeto de otra clase como parámetro** .
- Una relación de dependencia es una relación de “uso”. Un cambio en una cosa en particular puede afectar a otras cosas que la usan, y usar una dependencia cuando es necesario indicar que una cosa usa otra.
- Se indica la relación con una línea punteada con una punta de flecha.




Whaaaaat...

¿Cómo hacemos un diagrama de clase?

1. Identificar las clases
2. Definir los atributos y métodos
3. Definir las relaciones
4. Confeccionar el diagrama UML

Enunciado

“Una biblioteca presta libros a sus usuarios. Cada usuario puede pedir varios libros, y cada libro puede estar prestado a un solo usuario a la vez. Los libros tienen un título y un autor. Los usuarios tienen un nombre y un número de identificación. Cuando un usuario toma un libro prestado, se registra la fecha del préstamo y la fecha de devolución esperada”.

Enunciado

“Una biblioteca presta libros a sus **usuarios**. Cada usuario puede pedir varios **libros**, y cada libro puede estar prestado a un solo usuario a la vez. Los libros tienen un título y un autor. Los usuarios tienen un nombre y un número de identificación. Cuando un usuario toma un libro prestado, se registra la fecha del **préstamo** y la fecha de devolución esperada”.

Paso 1: Identificar las clases

- **Usuario** → Representa a una persona que puede pedir libros prestados.
- **Libro** → Representa los libros disponibles en la biblioteca.
- **Préstamo** → Representa la acción de prestar un libro.

Paso 2: Definir los atributos y métodos

- **Usuario**
  - **Atributos:** idusuario, nombre
  - **Métodos:** pedirLibro(), devolverLibro()

- **Libro**
  - **Atributos:** titulo, autor, disponibilidad
  - **Métodos**: reservar(), devolver()

- **Préstamo**
  - **Atributos:** fecha préstamo, fecha devolución
  - **Métodos:** registrarPrestamo(), registrarDevolucion()

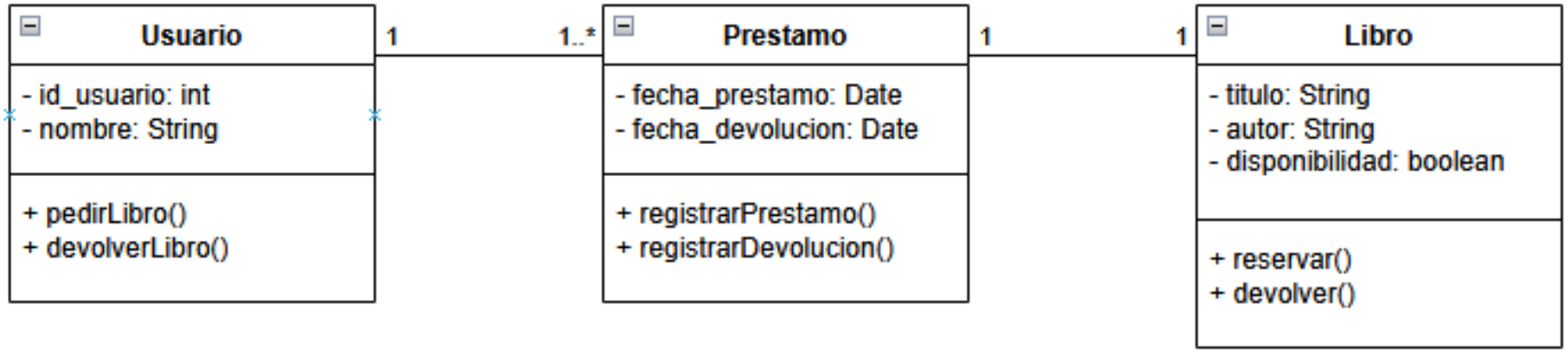
Paso 3: Definir las relaciones

- Relación entre **Usuario** y **Préstamo** → Asociación 1 a muchos
  - *Justificación*: un préstamo no puede existir sin un usuario, pero ambos pueden existir independientemente (un usuario puede existir sin haber tomado préstamos).

- Relación entre **Libro** y **Préstamo** → Asociación 1 a 1
  - *Justificación*: el libro y el préstamo pueden existir independientemente (un libro en la biblioteca no necesariamente está en préstamo).

- Relación entre **Usuario** y **Libro** → Asociación Indirecta
  - *Justificación*: no hay una relación directa entre ambas clases, sino que se relacionan a través de la clase préstamo. Esta relación no figura en el diagrama.

Paso 4: Confeccionar el diagrama UML



Tarea: diagramas de clase UML

Ejercicio 1

Una empresa de alquiler de vehículos necesita un sistema para gestionar su flota. Existen distintos tipos de vehículos, como autos y motos, los cuales comparten ciertas características como marca, modelo y año de fabricación, pero de los autos también se registran la cantidad de puertas y de las motos el tipo de manubrio. Cada vehículo puede ser alquilado por un cliente, del cual se registran su nombre, apellido, DNI y un número de licencia de conducir. Un cliente puede alquilar varios vehículos (entre dos fechas puntuales), pero cada vehículo solo puede estar alquilado a un cliente a la vez.

El sistema también debe arrojar cierta información útil: determinar si el vehiculo es antiguo (10 o más años); ver los detalles completos de cada moto y auto; ver el historial de alquiler de cada cliente de la empresa; y averiguar cuanto tiempo falta para que finalice un alquiler actual.

Tarea: diagramas de clase UML

Ejercicio 2

Una tienda en línea maneja pedidos realizados por clientes. Cada pedido puede contener múltiples productos, y un cliente puede realizar varios pedidos. Un producto tiene un código único, un nombre, un precio y una cantidad en stock. Cada pedido tiene una fecha y un estado (pendiente, enviado o entregado); este estado se puede modificar en cualquier momento, así como también obtener el precio total. Los productos están agregados en un pedido, lo que significa que un producto puede existir fuera del pedido y ser parte de múltiples pedidos. Los clientes se registran en el sistema con nombre, apellido e email, y además de realizar pedidos pueden ver el historial de sus pedidos previos.

Otras actividades propias del sistema incluyen actualizar el stock de cada producto cuando se modifiquen sus existencias y obtener un listado con la información completa de un producto