ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Diagrama de Clases

Autora: Rojas Córsico, Ivana Soledad

Los diagramas de clases son uno de los tipos de diagramas más útiles en UML porque representan claramente la estructura de un sistema o parte de él modelando las relaciones entre clases, atributos, operaciones y objetos.

Los diagramas de clases utilizan símbolos y diferentes notaciones para representar la estructura de clases. A continuación se muestran algunos de los símbolos y notaciones más comunes para ayudarle a comprender mejor los diagramas de clases.

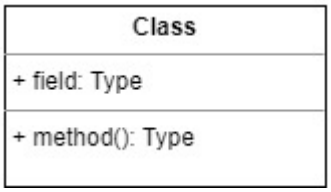
Símbolos y notaciones

A continuación se muestran algunos de los símbolos y notaciones del diagrama de clases más utilizados.

Clase

Una clase está representada por un rectángulo con tres compartimentos: nombre de clase, atributos (características) y operaciones (comportamiento). Un rectángulo de doble marco representa una clase abstracta de la que no se puede crear una instancia.

Ejemplo:



¿Cómo identifico las clases? Para identificar una clase, primero debemos entender qué es una clase. Una clase es una abstracción de un objeto del mundo real por lo que debemos identificar sus características esenciales (atributos) y su comportamiento (métodos). Una vez que identificamos sus atributos y métodos podemos modelar usando la representación de arriba.

Modificadores de acceso en atributos y métodos

Todas las clases poseen distintos niveles de acceso en función del modificador de acceso (visibilidad). Para modelar, utilizamos los siguientes símbolos:

- **Público (+).** Cuando se utiliza el modificador "public", el miembro al que se aplica es visible y accesible desde cualquier parte del programa.
- **Privado (-).** Cuando se utiliza el modificador "private" se restringe el acceso al miembro al interior de la clase en la que se encuentra. Es decir, solo los métodos y variables dentro de la misma clase pueden acceder a él.

- **Protegido (#).** Cuando se utiliza el modificador "protected" permite que el miembro sea accesible desde la misma clase y desde las subclases (clases hijas) de la clase en la que se encuentra.

¿Cómo identifico el modificador de acceso? Los modificadores de acceso son fundamentales en la programación orientada a objetos porque nos permiten gestionar y controlar la visibilidad y el acceso a los miembros de la clase. Esto nos ayuda a mantener un código más seguro, modular y fácil de mantener. Además los mismos permiten establecer una interfaz clara y bien definida para interactuar con el resto de clases y objetos. Por lo tanto, hay que analizar la visibilidad de los miembros de clase uno a uno.

¿Cuándo utilizar el modificador privado? El modificador de acceso privado se utiliza para restringir el acceso a un atributo o método de clase. Cuando el atributo o método de clase se declara como “privado”, sólo puede ser accedido dentro de la propia clase que es declarado. Esto significa que otros objetos o clases no podrán acceder directamente a él.

A continuación, algunas situaciones recomendable para utilizar el modificador de acceso privado:

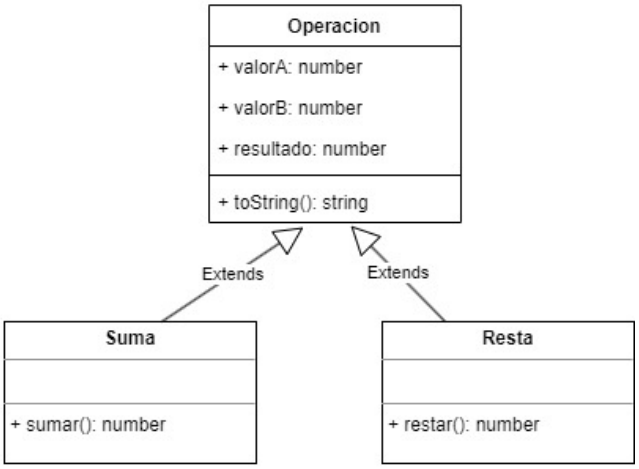
1- **Encapsulamiento de datos:** Declarar los atributos de una clase como privados evita que otros objetos o clases modifiquen directamente esos atributos. En su lugar, debe acceder a él a través de métodos getter y setter para poder controlar cómo se accede y modifica la información.
2- **Protección de datos sensibles:** Si tienes un atributo que contiene información confidencial, como una contraseña o un número de tarjeta de crédito, se recomienda declarar el atributo como privado para que no se pueda acceder a él desde otras partes de su código.
3- **Control de la lógica interna de una clase:** Declarar un método en una clase como "privado" garantiza que el método solo sea utilizado internamente por la propia clase. Esto ayuda a mantener la lógica interna coherente y coherente y evita que otros objetos o clases modifiquen directamente la lógica interna.
4- **Prevención de modificaciones no deseadas:** Declarar atributos de una clase como privados, previene que otras clases ya subclases puedan volver a sobreescribir los atributos. Esto ayuda a mantener la integridad del código.

¿Cuándo utilizar el modificador protegido? El modificador de acceso “protegido” se utiliza cuando queremos que un miembro de una clase sea accesible desde sus clases hijas o subclases.

A continuación, algunas de las situaciones recomendables para utilizar el modificador de acceso protegido:

Herencia: Si tienes una superclase o clase padre y se desea que los miembros de esa clase sean accesibles para sus subclases o clases hijas, se recomienda utilizar el modificador de acceso protegido. Esto permite que las subclases accedan a estos miembros y los utilicen en sus propios métodos.

Ejemplo:



¿Cuándo utilizar el modificador de acceso público? El modificador de acceso público se utiliza para permitir el acceso a atributos y métodos desde cualquier parte del programa. Cuando utilizamos este modificador, estamos indicando que el miembro es público y puede ser accedido sin ninguna restricción.

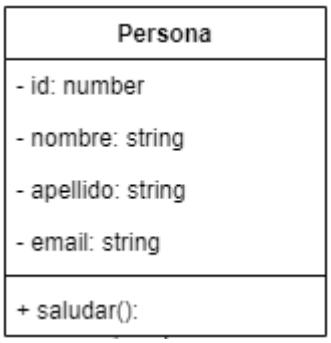
A continuación, algunas de las situaciones recomendables para utilizar el modificador de acceso público:

1- Cuando necesitas acceder a un atributo o método desde fuera de la clase. Esto es útil cuando requieres compartir información entre objetos o clases.
2- Cuando requieres que los atributos y métodos sean accedidos desde otras clases. Esto es útil cuando estamos creando librerías que serán utilizadas por otros programadores.

Tipos de datos

Todos los atributos y métodos deben especificar el tipo de datos. Se puede modelar con tipos de datos primitivos, como enumeraciones, arreglos y otros.

Ejemplo:



Interfaces

Contrato que la clase que la implemente debe cumplir. Son similares a una clase, excepto por que una clase puede tener una instancia de su tipo, y una interfaz no (a menos que se implemente mediante polimorfismo)

Objetos

Instancias de una clase o clases. Los objetos se pueden agregar a un diagrama de clases para representar instancias prototípicas o concretas. Se representan mediante una elipse.

Relaciones entre clases

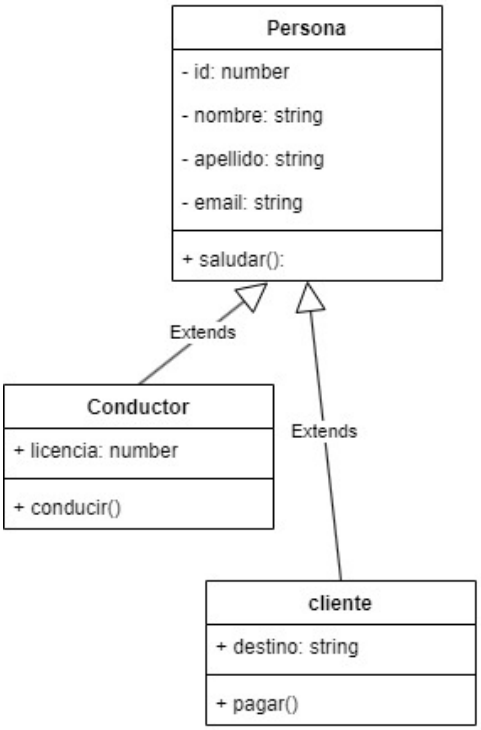
Las relaciones entre clases se pueden representar mediante líneas, flechas y otros símbolos que indican la propiedad o dependencia de la clase.

Algunos de los principales tipos de relaciones entre clases son:

Herencia (Generalización/Especialización “es un”)

Es una relación en la que una subclase o clase derivada recibe la funcionalidad de una superclase o clase principal, también se conoce como "generalización". Se simboliza mediante una línea de conexión recta con una punta de flecha cerrada que señala a la superclase.

Ejemplo:

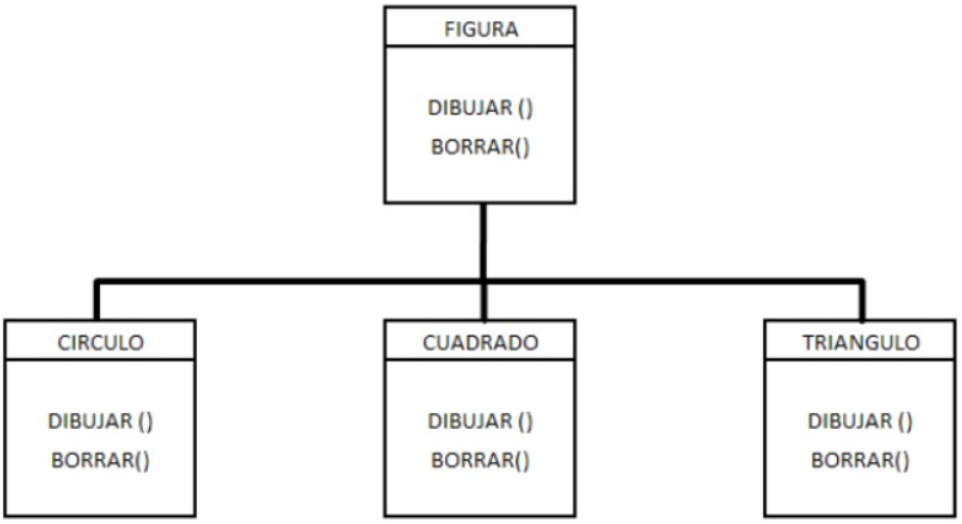


¿Cómo identificar una relación de herencia? La herencia permite la definición de un nuevo objeto a partir de otro, agregando nuevos atributos y/o métodos o sobrescribiendo los mismos. Evita la repetición de código y permite la reusabilidad. Se trata de una relación de generalización / especialización o “es un”

- **La generalización**. Es un mecanismo de abstracción mediante el cual un conjunto de clases de objetos son agrupados en una clase de nivel superior (Superclase), donde las semejanzas de las clases constituyentes (Subclases) son enfatizadas, y las diferencias entre ellas son ignoradas.

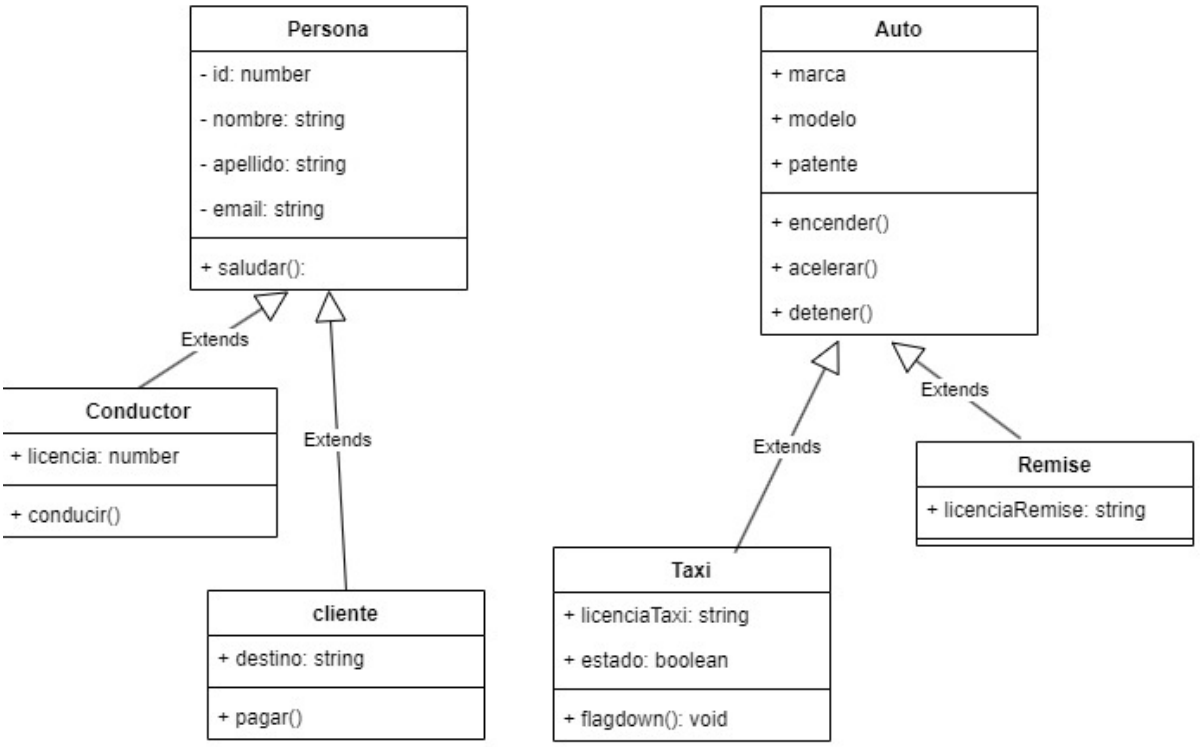
  En consecuencia, a través de la generalización:
  - La superclase almacena datos generales de las subclases
  - Las subclases almacenan sólo datos particulares.

  Ejemplo:



- **La especialización.** Es un mecanismo de abstracción mediante el cual el conjunto de clases heredan de la clase de nivel superior (superclase) atributos y métodos pero incorporan atributos y métodos propios.

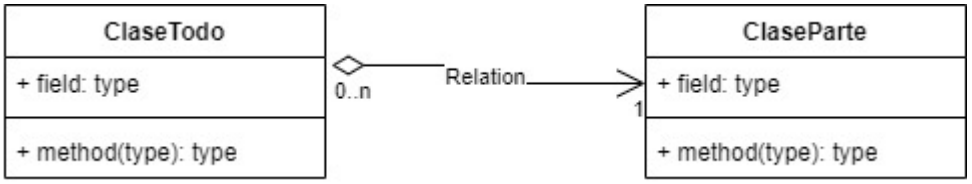
  Ejemplo:



Agregación (Todo/Parte “forma parte de”)

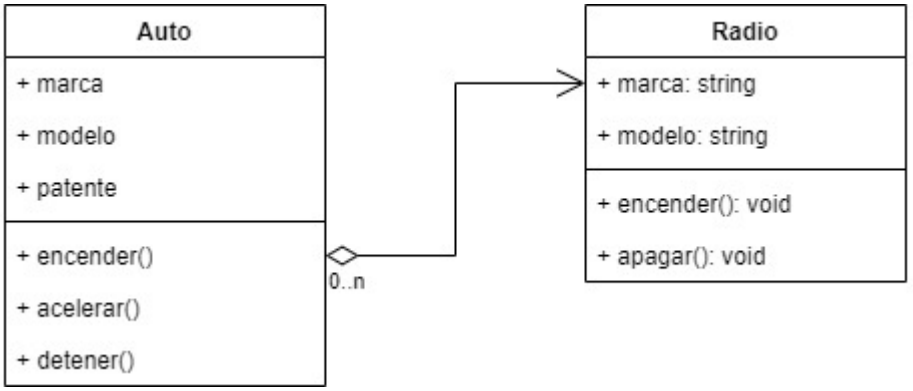
Es una relación que representa a un objeto compuesto por otros. El objeto del nivel superior es el todo, mientras que los objetos de los niveles inferiores son partes.

La relación forma parte de, se representa como sigue:



Nota que el diamante va pegado en la ClaseTodo.

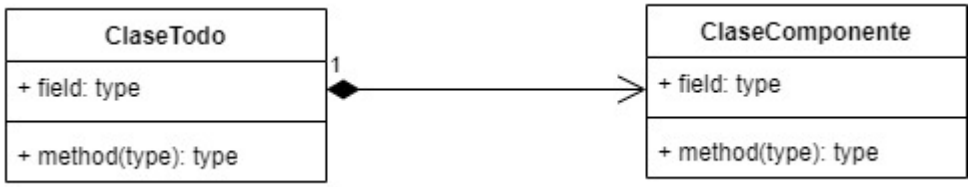
Ejemplo:



Composición (Todo/Parte “es parte elemental de”, “está compuesto por”).

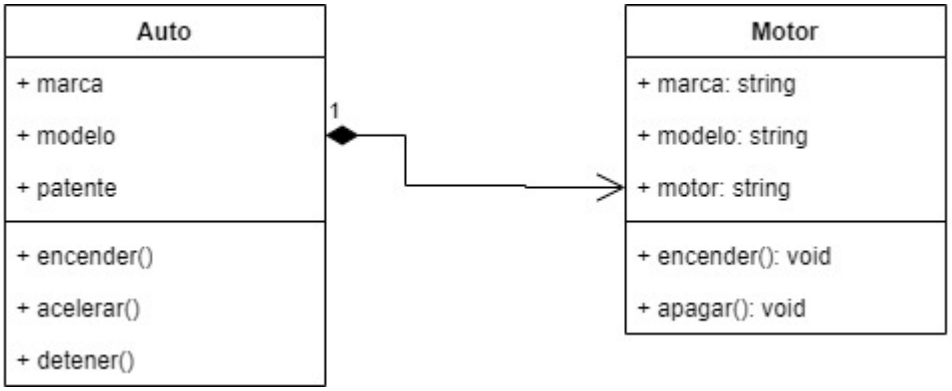
Es una relación que representa un objeto compuesto por otro pero posee una relación más fuerte que la agregación dado que, si el componente es eliminado o desaparece, la clase todo deja de existir o no funciona como tal.

La relación “es parte elemental de”, se representa como sigue:



Nota que el diamante va pegado en la ClaseTodo.

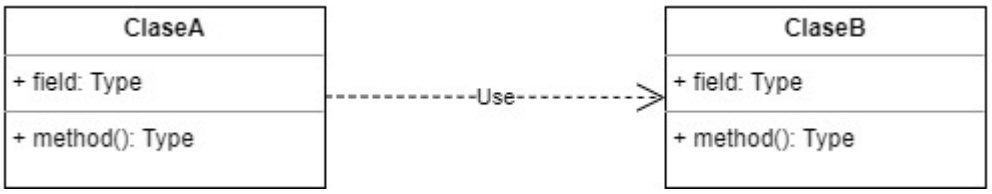
Ejemplo:



Asociación (“usa a”)

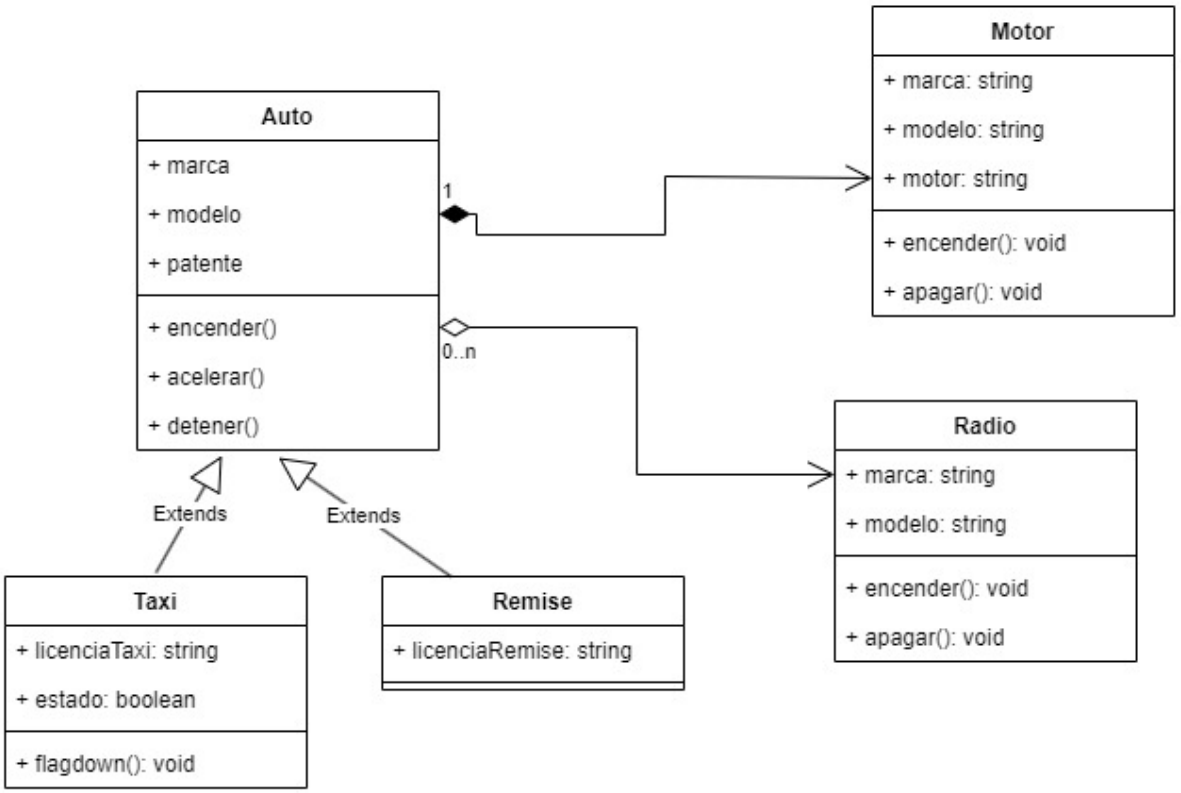
Es una relación que se establece cuando dos clases tienen una dependencia de utilización, es decir, una clase utiliza atributos o métodos de la otra para funcionar. Estas clases no necesariamente están en jerarquía, es decir que no necesariamente una es superclase y la otra subclase.

La relación “usa a”, se representa como sigue:



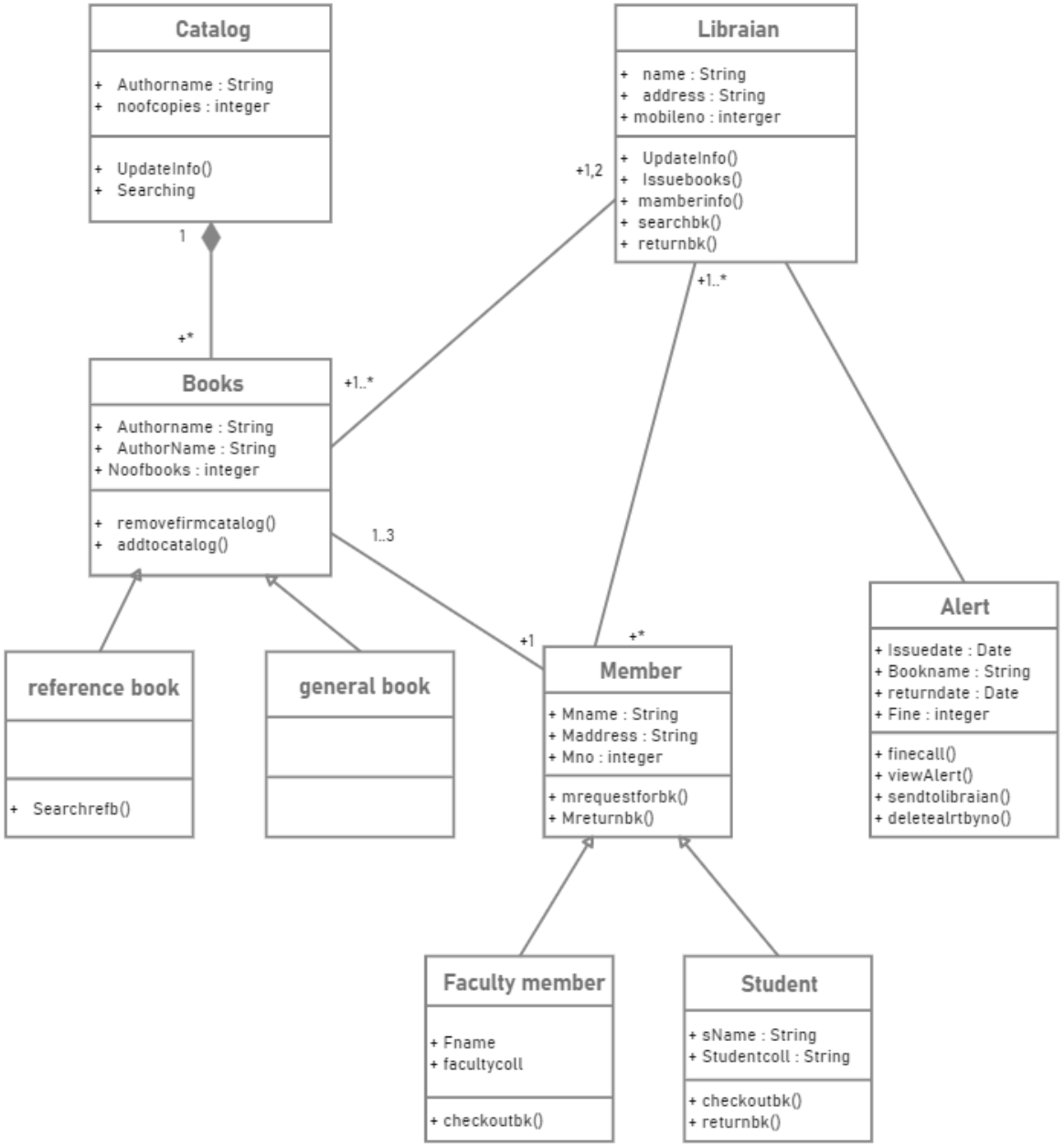
Dónde la flecha especifica la dirección de uso. En este caso, la clase A usa algún método o atributo de la ClaseB.

Finalmente y en base a los ejemplos previos, nuestro diagrama de clases:


Motor
+ marca: string
+ modelo: string
+ motor: string
+ encender(): void
+ apagar(): void
Auto
+ marca
+ modelo
+ patente
+ encender()
+ acelerar()
+ detener()
1
0..n
Radio
+ marca: string
+ modelo: string
+ encender(): void
+ apagar(): void
Extends
Extends
Taxi
+ licenciaTaxi: string
+ estado: boolean
+ flagdown(): void
Remise
+ licenciaRemise: string

Ejemplos de Diagramas de Clases

Diagrama de Clases de una librería



Fuente: https://www.edrawsoft.com/es/example-uml-class-diagram.html