Programación 2

Conceptos POO. UML

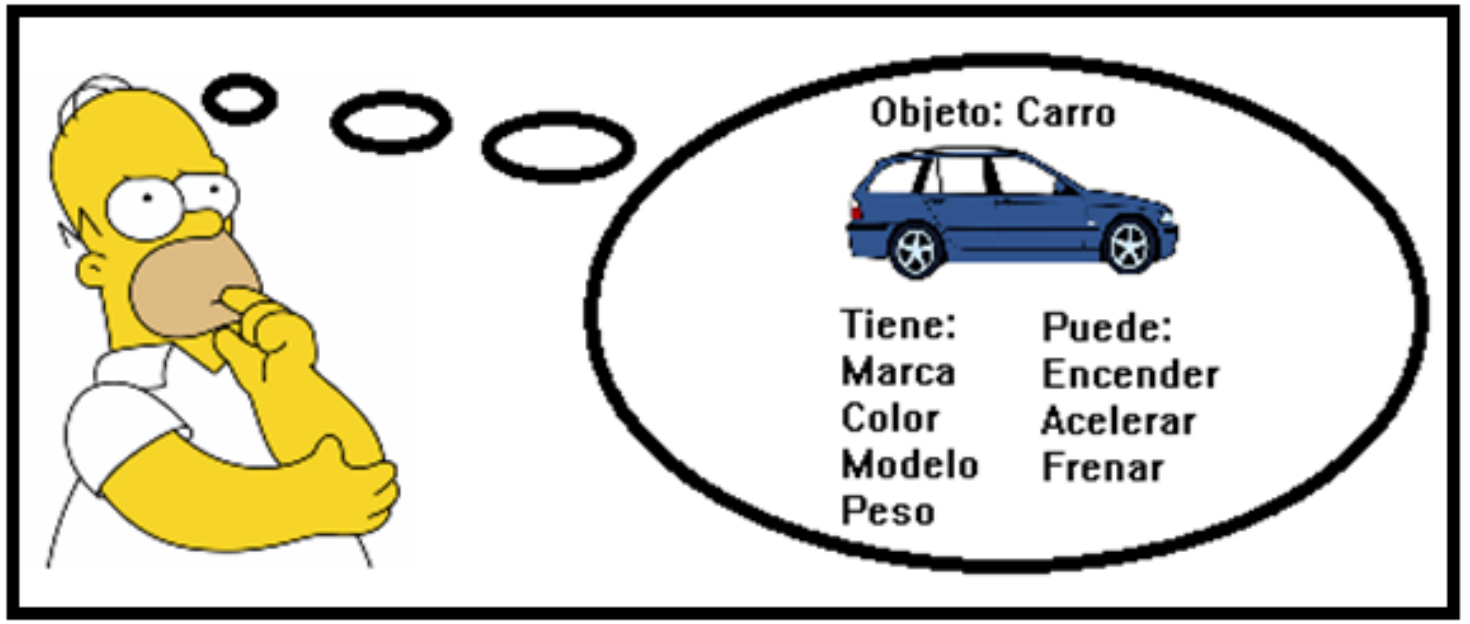
¿Qué es la POO?

En el paradigma imperativo usamos el concepto de llamadas de procedimientos para modularizar el código. Estas funciones y procedimientos simplemente contienen una serie de líneas de instrucciones a llevar a cabo. Suele relacionarse con un enfoque top-down de división del problema.

En la **Programación Orientada a Objetos** se busca tratar a toda entidad como si fuera un **objeto** en la vida real, donde cada objeto tiene algunas **propiedades** y **funcionalidades** propias, es decir, ***en la POO se busca encapsular datos y comportamiento dentro de los objetos.***

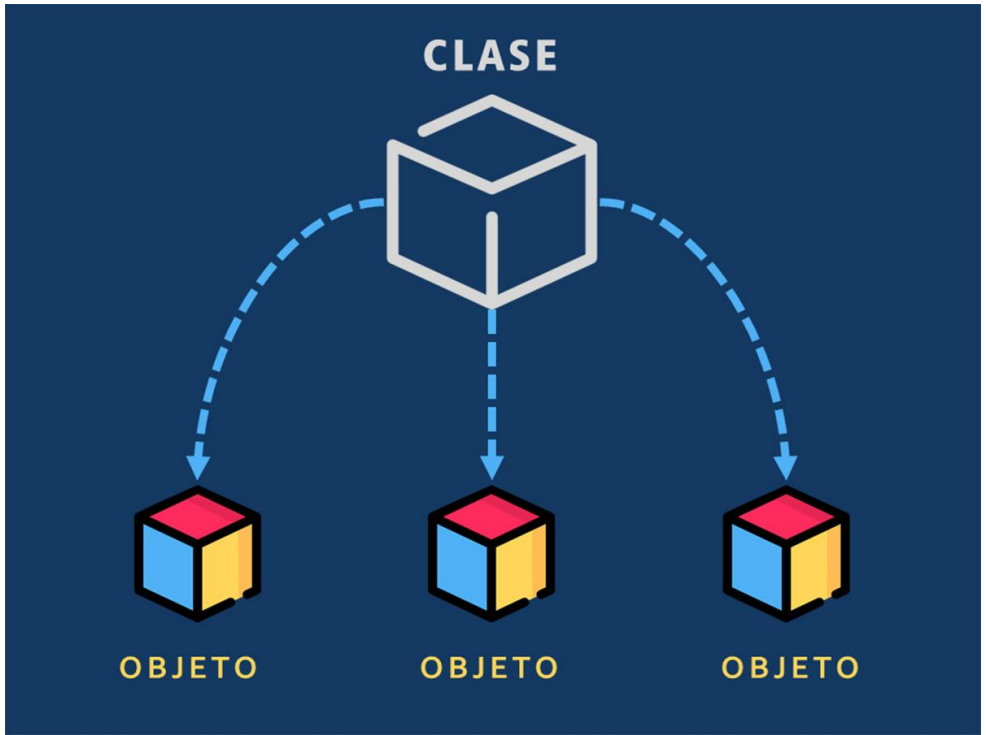
Programación Orientada a Objetos

- En POO creamos objetos que tienen ***atributos*** similares a los objetos en la vida real.
- Los objetos en la vida real tienen algunas propiedades y pueden realizar algunas funciones.



Clase y objeto

- Una clase es una plantilla (un molde) para crear objetos.
- Los objetos tienen atributos (características) y métodos (funciones) asociados.



Explicación según Steve Jobs (entrevista 1994)

¿Podría explicarnos, en términos sencillos, qué es exactamente el software orientado a objetos?

“Los objetos son como las personas. Son seres vivos y respirables que tienen en su interior conocimiento sobre cómo hacer las cosas y tienen memoria en su interior para poder recordarlas. Y en lugar de interactuar con ellos en un nivel muy bajo, interactúas con ellos en un nivel muy alto de abstracción, como lo estamos haciendo aquí.

Aquí tienes un ejemplo: si soy tu objeto de lavandería, puedes darme tu ropa sucia y enviarme un mensaje que diga: "¿Puedes lavarme la ropa, por favor?". Resulta que sé dónde está la mejor lavandería de San Francisco. Y hablo inglés y tengo dólares en mis bolsillos. Así que salgo, paro un taxi y le digo al conductor que me lleve a este lugar en San Francisco. Voy a lavar tu ropa, vuelvo al taxi y vuelvo aquí. Te doy tu ropa limpia y te digo: “Aquí tienes tu ropa limpia”.

No tienes idea de cómo hice eso. No tienes conocimiento del lugar de lavado. Quizás hablas francés y ni siquiera sabes parar un taxi. No puedes pagar por uno, no tienes dólares en el bolsillo. Sin embargo, sabía cómo hacer todo eso. Y no era necesario que supieras nada de eso. Toda esa complejidad estaba escondida dentro de mí y pudimos interactuar en un nivel muy alto de abstracción. Eso es lo que son los objetos. Encapsulan la complejidad y las interfaces con esa complejidad son de alto nivel.”

fuente: https://www.rollingstone.com/culture/culture-news/steve-jobs-in-1994-the-rolling-stone-interview-231132/

Clases y objetos - Terminología

Clase: es un "*molde*" que define qué datos tendrá un objeto (propiedades, atributos) y define qué acciones puede realizar (comportamiento, métodos). En POO, la estructura de datos y los métodos de cada tipo de objeto se manejan juntos. No se puede tener acceso o control de la estructura de datos excepto mediante los métodos que forman parte del tipo de objeto.

Objeto: se refiere a una instancia particular de la clase. Refiere a los datos particulares y a los métodos mediante los cuales se controlan a los propios datos. Estos datos definen un estado, y los métodos definen su comportamiento.

Programación Orientada a Objetos - Pilares

La Programación orientada a objetos es un paradigma de programación que utiliza "objetos" y sus interacciones para diseñar aplicaciones y programas informáticos.

Los **pilares fundamentales** de la POO son:

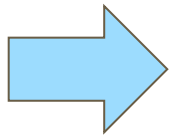
- **Abstracción**
- **Encapsulamiento**
- **Herencia**
- **Polimorfismo**

Cada uno de estos conceptos juega un papel crucial en la forma en que se estructura y se desarrolla el software orientado a objetos, en el transcurso de la materia los desarrollaremos.

Recordando: fases Análisis de requerimientos y Diseño

Análisis

- ¿Qué? objetivos debe perseguir el sistema.
- Se definen los elementos implicados: usuarios, dispositivos, otros sistemas.
- Qué problema hay que resolver.
- Qué características debe tener la solución y qué restricciones debe cumplir



Diseño

- ¿Cómo? alcanzará el sistema esos objetivos.
- Se definen los módulos que integrarán el sistema y la forma en que se relacionan entre sí.
- Se crea un modelo de los elementos implicados (UML: Unified Modeling Language)

Diseño de clases

Durante el análisis de requerimientos, el analista construye un diagrama de clases que modela el problema. El diseñador del sistema completa el diagrama para modelar la solución.

Un **diagrama de clases** es una representación visual que permite modelar la estructura de un sistema de software a partir de las clases que lo componen.

Se construye usando un lenguaje de modelado.

Este diagrama de clases que resulta como producto de la etapa de análisis de requerimientos, representa el modelado del problema.

Modelado del problema

Un modelo es una representación que permite describir, explicar, analizar o simular un sistema o proceso a partir de sus entidades relevantes y el modo en que se relacionan.

Modelado del problema - Abstracción

La **abstracción** es el mecanismo fundamental que se utiliza para generar el modelo.

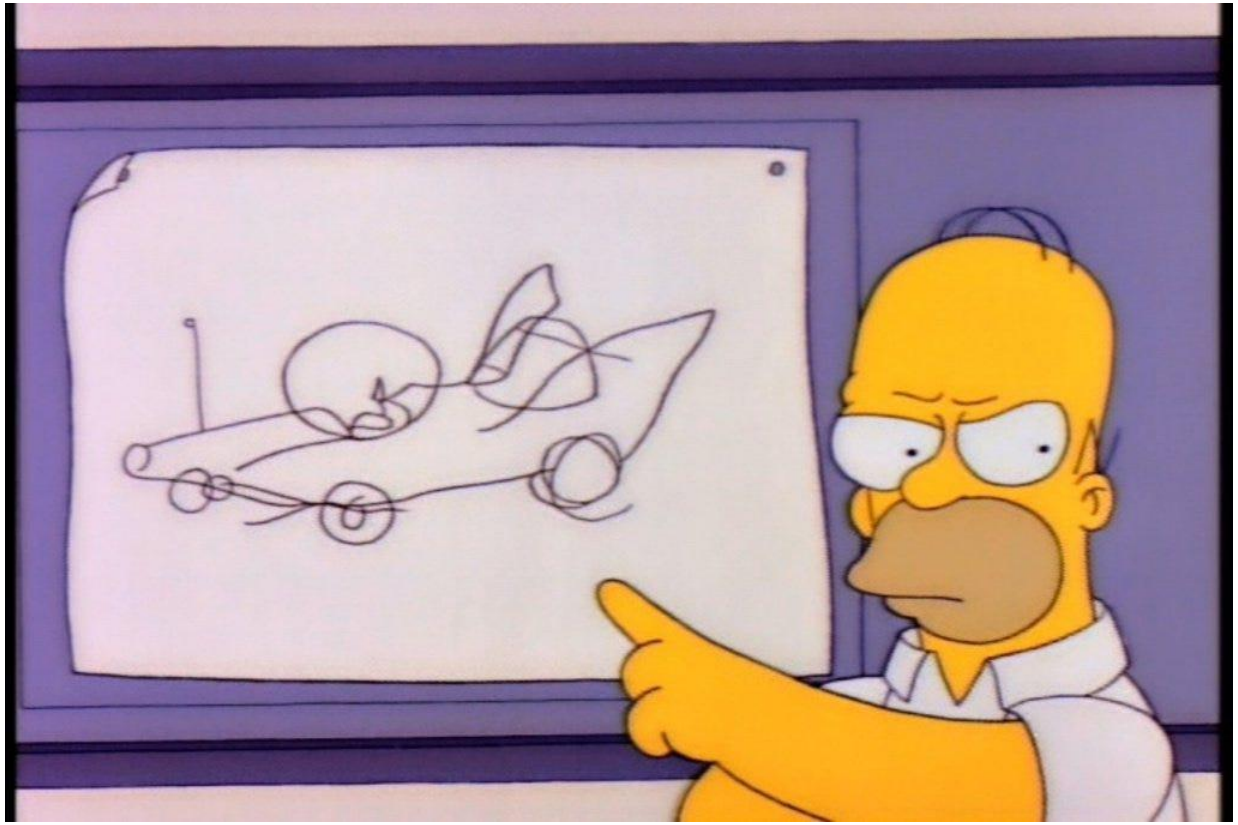
Una abstracción se genera a partir de una serie de operaciones mentales que permiten **identificar** entidades y caracterizarlas de acuerdo a sus propiedades y al modo en que interactúan con los demás. Una misma entidad se puede representar de distintas formas según las propiedades que se consideren esenciales y el nivel de detalle que se pretenda en la representación.

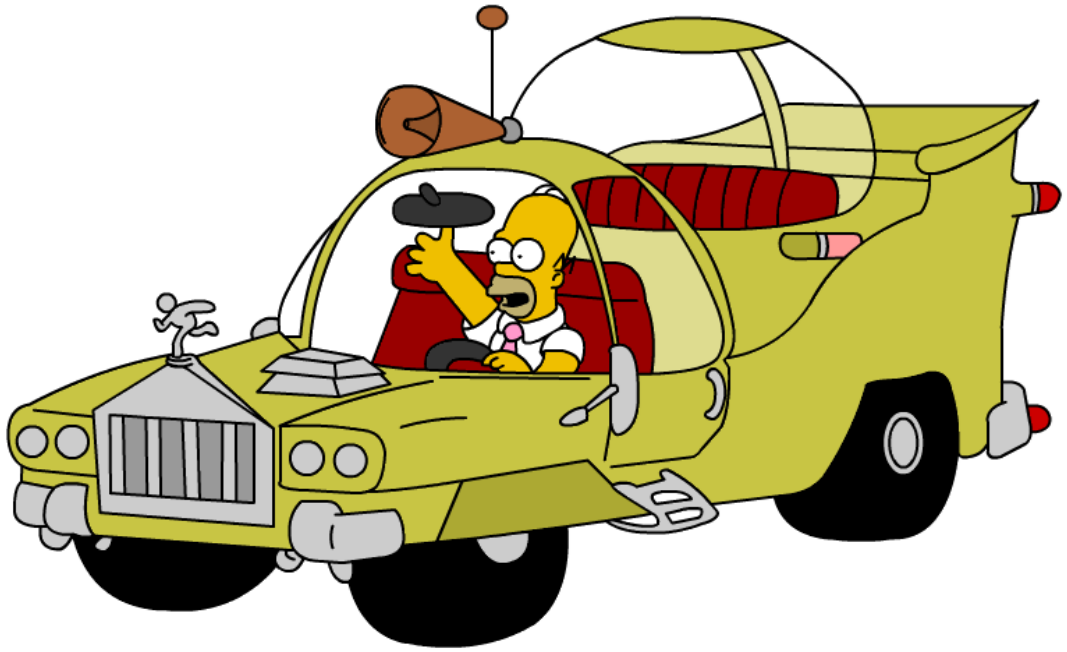
Es el proceso de simplificar un problema complejo enfocándose solamente en los aspectos relevantes para la solución

En el desarrollo de software esto significa centrarse en lo que es y hace un objeto, antes de decidir cómo debería ser implementado

Abstracción

Énfasis en el ¿Que hace?





Caso de estudio - librería

Imaginemos una tienda de libros. En este escenario, podemos identificar varios objetos con características y comportamientos comunes que nos permitirán comprender los conceptos de clases y objetos en la programación orientada a objetos (POO).

Objetos del problema:

- Libros
- Clientes
- Empleados

Caso de estudio - librería

Objeto del problema: **Libro**

Un objeto que representa un libro físico con las siguientes características:

Atributos: Título, autor, ISBN, género, precio, número de páginas, editorial, stock (cantidad disponible).

Comportamientos: Ser prestado, ser vendido, ser reservado, tener su información mostrada (título, autor, etc.).

Caso de estudio - librería

Objeto del problema: **Cliente**

Un objeto que representa a una persona que compra o interactúa con la tienda.

Atributos: Nombre, apellido, dirección, número de teléfono, correo electrónico, historial de compras (lista de libros comprados).

Comportamientos: Comprar libros, realizar reservas, consultar información sobre libros, recibir recomendaciones.

Caso de estudio - librería

Objeto del problema: **Empleado**

Un objeto que representa a un trabajador de la tienda.

Atributos: Nombre, apellido, cargo (vendedor, cajero, gerente), horario de trabajo, salario.

Comportamientos: Vender libros, atender clientes, gestionar el stock, realizar pedidos a proveedores.

Concepto de objeto

Un objeto del problema es una entidad, física o conceptual, que puede ser modelada a través de sus **atributos** y su **comportamiento**.

Cada **objeto del problema** en ejecución quedará asociado a un **objeto de software**.

Un objeto de software es un modelo, una representación abstracta de un objeto del problema.

Un objeto de software tiene una **identidad** y mantiene los valores de los atributos en su **estado interno**.

El **comportamiento** queda determinado por un conjunto de **servicios** (funcionalidades que ofrece a otros objetos o al sistema en general) y un conjunto de **responsabilidades** (obligaciones o tareas que el objeto debe cumplir, incluyendo la gestión de su estado interno, validaciones de datos, etc).

Concepto de Clase

Los objetos del problema pueden agruparse en **clases** de acuerdo a sus **atributos** y **comportamiento**. (clasificación)

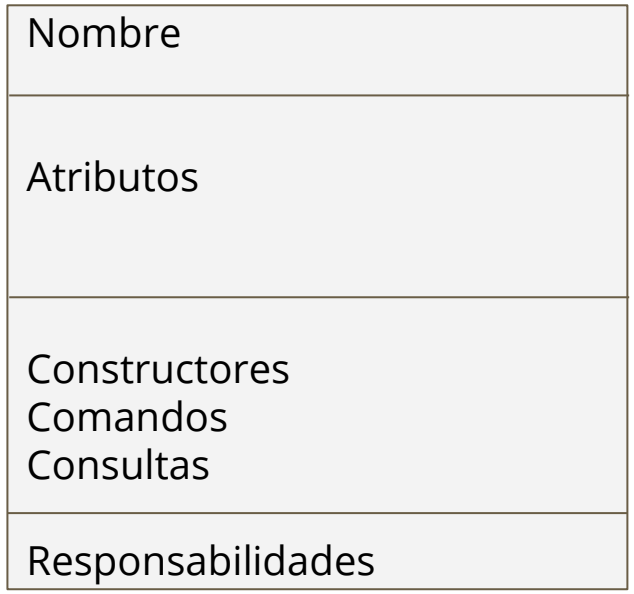
Todos los objetos de una clase van a estar modelados por un mismo conjunto de atributos y un mismo comportamiento.

Desde el punto de vista del diseño una clase es un patrón que establece los atributos y el comportamiento de un conjunto de objetos.

Un sistema de mediana escala puede modelarse a través de cientos o incluso miles de clases relacionadas entre sí.

Diagrama de una clase

En la etapa de **diseño** cada clase se modela mediante un diagrama:



Los servicios provistos por una clase pueden ser **constructores**, **comandos** o **consultas**.

El diagrama puede incluir notas o comentarios que describen **restricciones** o la **funcionalidad de los servicios**.

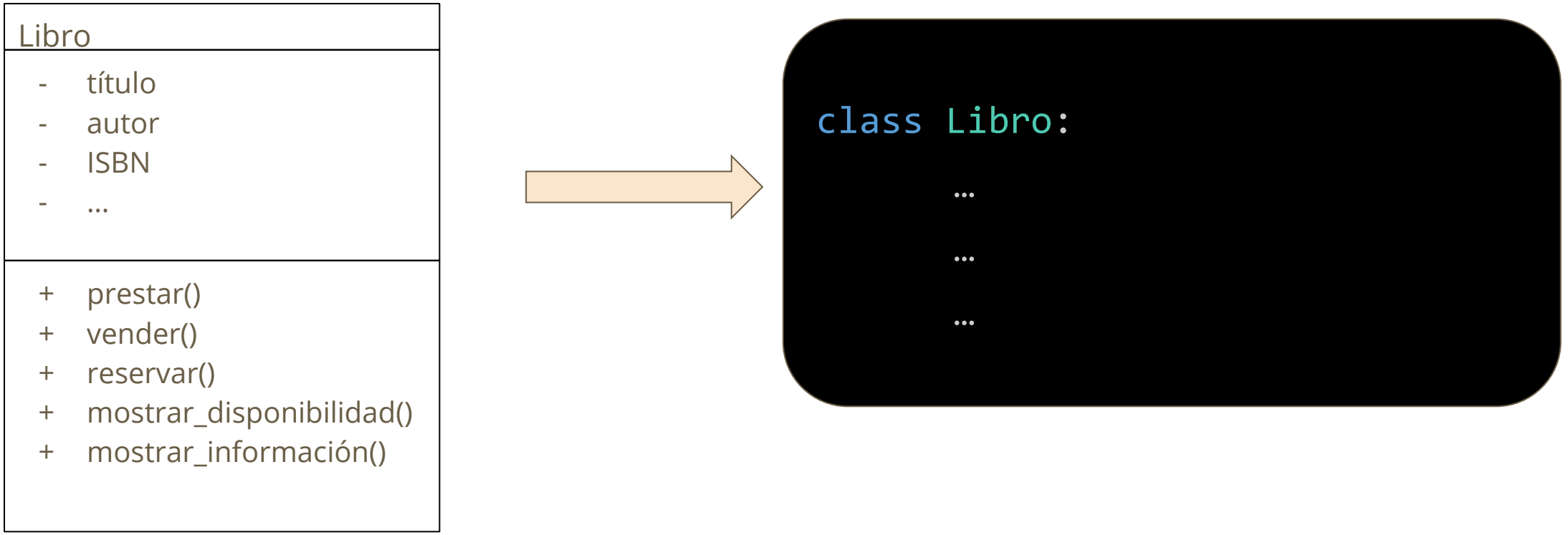
Clase

- El **nombre** de una clase representa al conjunto de instancias. Suele ser un sustantivo o un nombre representativo. Debe seleccionarse cuidadosamente.
- Los **atributos** son propiedades o cualidades que caracterizan a los objetos de la clase.
  Podemos clasificarlos en atributos de clase o atributos de instancia.
  Los atributos de clase son “globales en la clase”, comparten su valor con todas las instancias.
  Los atributos de instancia son valores que varían en cada objeto.
- Un **servicio** es una operación que todas las instancias de la clase pueden brindar.
  Los servicios pueden ser **métodos** o **constructores**.
  Los métodos pueden ser **comandos** o **consultas**.
- La **responsabilidad** representa un compromiso para la clase o un requerimiento.

El conjunto de servicios y responsabilidades determinan el comportamiento de las instancias de una clase.

Clase

En la etapa de **implementación** cada clase es una unidad de código, cada clase modelada en lenguaje de modelado va a corresponderse a una clase implementada en lenguaje de programación. :



Clase y objeto - Ejemplo

Clase

Libro
- título - autor - ISBN - género - precio - cantidad de páginas - editorial - stock
+ prestar() + vender() + reservar() + mostrar_disponibilidad() + mostrar_información()

Objetos

- Título: El Señor de los Anillos (Edición en estuche)
- Autor: J.R.R. Tolkien
- ISBN: 9788420667298
- Género: Fantasía épica
- Precio: $3.499
- Páginas: 1136
- Editorial: Minotauro
- stock: 200

- título: El Hobbit
- Autor: J.R.R. Tolkien
- ISBN: 9788420667067
- Género: Fantasía
- Precio: $16.95
- Páginas: 320
- Editorial: Minotauro
- stock: 106

- título: El Silmarillion
- Autor: J.R.R. Tolkien
- ISBN: 9788420667281
- Género: Fantasía
- Precio: $22.95
- Páginas: 384
- Editorial: Minotauro
- stock: 254

Modelado del problema - Clasificación

La **clasificación** es otro mecanismo fundamental para la construcción de modelos.

Una clasificación se genera a partir de una serie de operaciones mentales que permiten **agrupar** entidades en función de sus semejanzas y diferencias.

El criterio de clasificación va a estar ligado a las propiedades que se identificaron como relevantes.

Diagrama de clases - UML

El **diagrama de clases** es uno de los diagramas de **estructura** de UML y se utiliza para representar los elementos que componen un sistema de información desde un **punto de vista estático**.

El diagrama de clases es un diagrama puramente orientado al modelo de programación orientado a objetos, ya que define las clases que se utilizarán cuando se pase a la fase de implementación y la manera en que se relacionan las mismas.

Elementos del diagrama:

- **Clases**
- **Relaciones**
- interfaces

Diagrama de clases UML - Clases

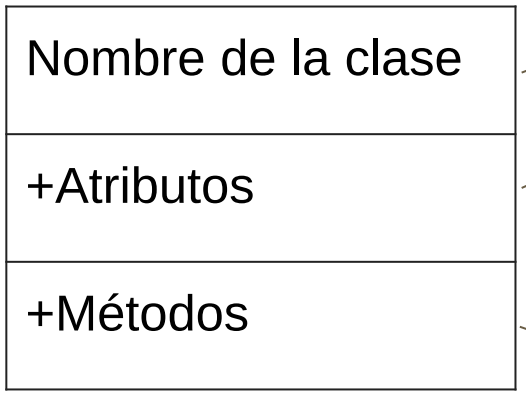
Las clases son el elemento principal del diagrama y representa, como su nombre indica, una clase dentro del paradigma de la orientación a objetos. Este tipo de elementos normalmente se utilizan para representar conceptos o entidades del «negocio».

Una clase define un grupo de objetos que comparten características, condiciones y significado.

En el diagrama, una clase está compuesta por tres elementos: **nombre** de la clase, **atributos** y **funciones**. Estos elementos se incluyen en la representación (o no, dependiendo del nivel de análisis).

Diagrama de clases UML - Clases

Para representar la clase con estos elementos se utiliza una caja que es dividida en tres zonas utilizando para ello líneas horizontales:



Primer sección: **nombre de la clase**

Segunda sección: atributos.
visibilidad nombre_atributo : tipo = valor-inicial { propiedades }
Si bien esa es la notación “oficial” de escribirlos, es común simplificando únicamente poniendo el nombre y el tipo o únicamente el nombre.

Tercera sección: **funciones** que ofrece la clase. De forma parecida a los atributos, sigue el siguiente formato “oficial”:
visibilidad nombre_funcion {parametros} : tipo-devuelto { propiedades }

Diagrama de clases UML - Clases

Tanto los atributos como las funciones incluyen al principio de su descripción la visibilidad que tendrá. Esta visibilidad se identifica escribiendo un símbolo y podrá ser:

- (**+**) Pública. Representa que se puede acceder al atributo o función desde cualquier lugar de la aplicación.
- (**-**) Privada. Representa que se puede acceder al atributo o función únicamente desde la misma clase.
- (**#**) Protegida. Representa que el atributo o función puede ser accedida únicamente desde la misma clase o desde las clases que hereden de ella (clases derivadas).

Diagrama de clases UML - Clases

Ejemplo de clase:

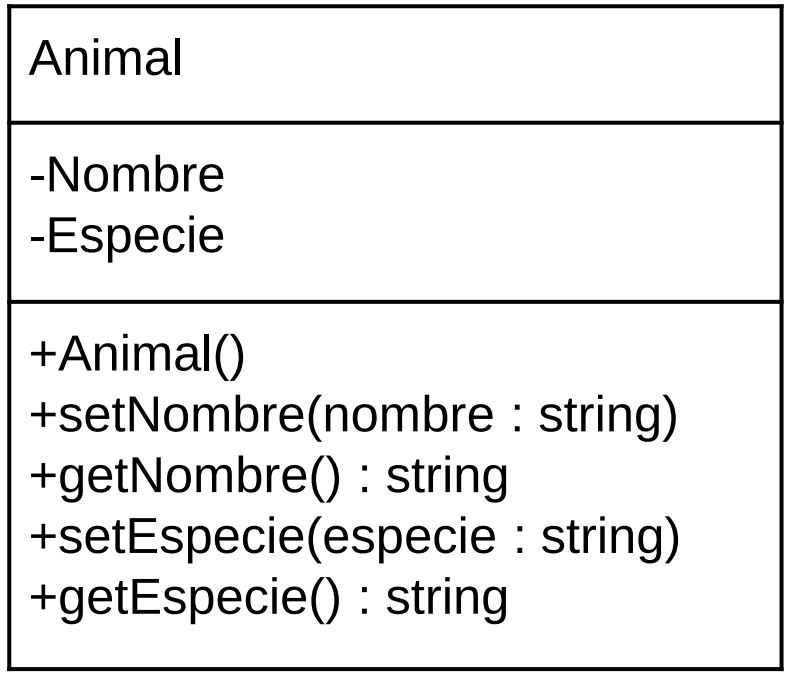


Diagrama de clases UML - Relaciones

Una relación identifica una interacción entre clases. Esta relación puede ser entre dos o más clases (más común) o una clase hacía sí misma (menos común, pero existen), este último tipo de dependencia se denomina dependencia reflexiva.

Las relaciones se representan con una línea que une las clases, esta línea variará dependiendo del tipo de relación.

Las relaciones tienen propiedades (multiplicidad, tipo y nombre), que dependiendo la profundidad que se quiera dar al diagrama se representarán o no.

Diagrama de clases UML - Relaciones

Multiplicidad. Se refiere al número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango. Se utiliza ***n*** o ***** para identificar un número cualquiera.

Nombre de la asociación. En ocasiones se escribe una indicación de la asociación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como por ejemplo: «***Una empresa contrata a n empleados***»

Diagrama de clases UML - Tipos de Relaciones

Un diagrama de clases incluye los siguientes tipos de relaciones:

- Asociación.
- Agregación.
- Composición.
- Dependencia.
- Herencia.

Diagrama de clases UML - Relaciones de asociación

Este tipo de relación es el más común y se utiliza para representar **dependencia semántica**.

Se representa con una simple línea continua que une las clases que están incluidas en la asociación.

Un ejemplo de asociación podría ser: «*Una mascota pertenece a una persona*».

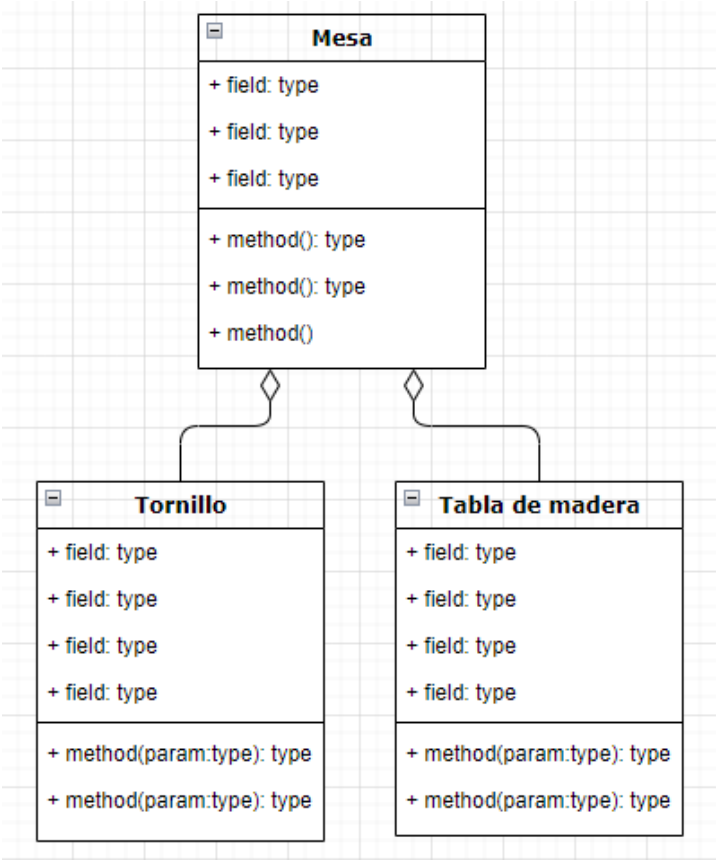
Diagrama de clases UML - Relaciones de agregación

Es una **representación jerárquica** que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que **un objeto es parte de otro**, pero aun así **debe tener existencia en sí mismo**.

Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

Un ejemplo de esta relación podría ser: «*Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa*».

Los tornillos y las tablas de madera podrían formar parte de más objetos, por lo que interesa especialmente su abstracción en otra clase.



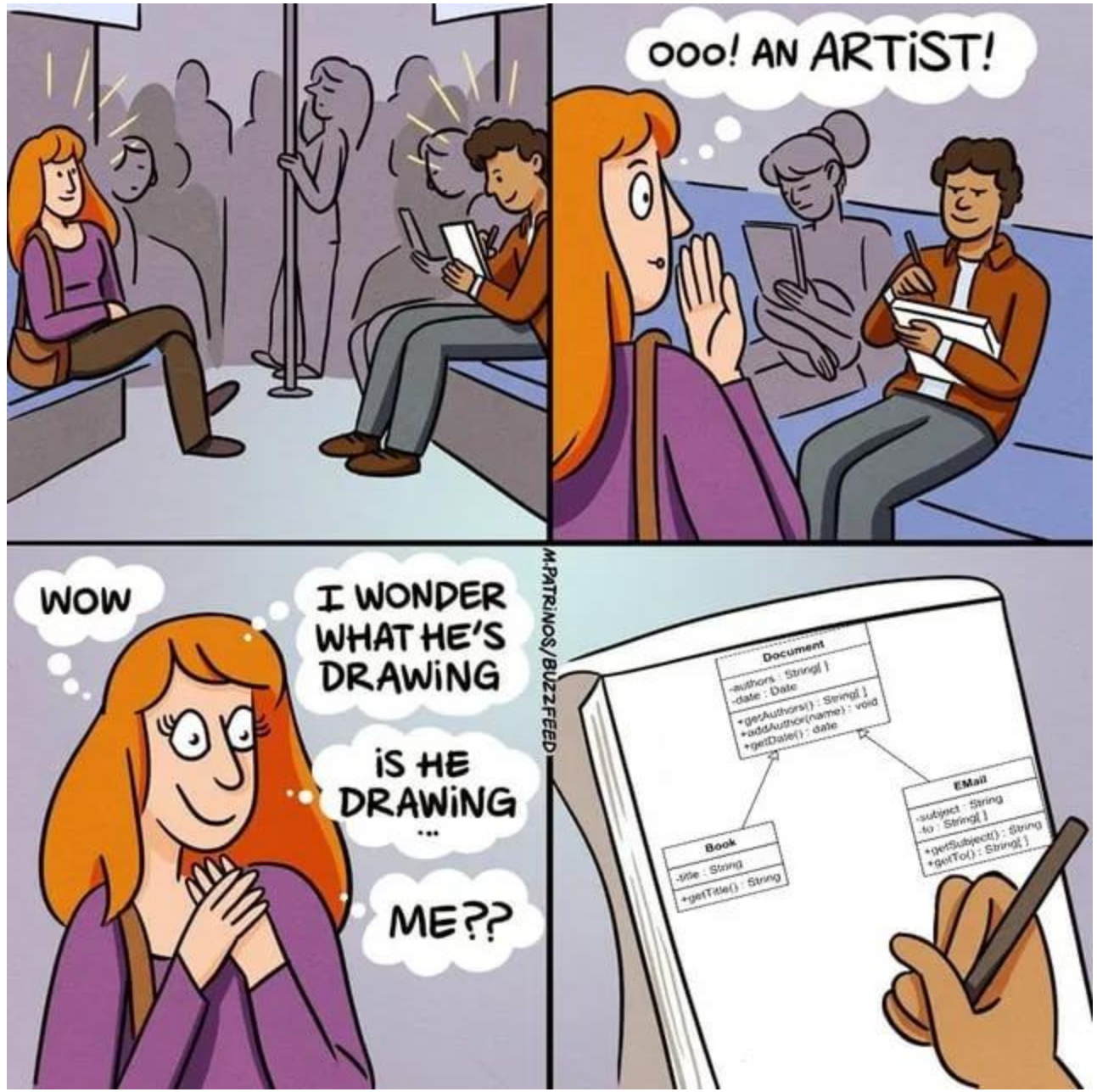
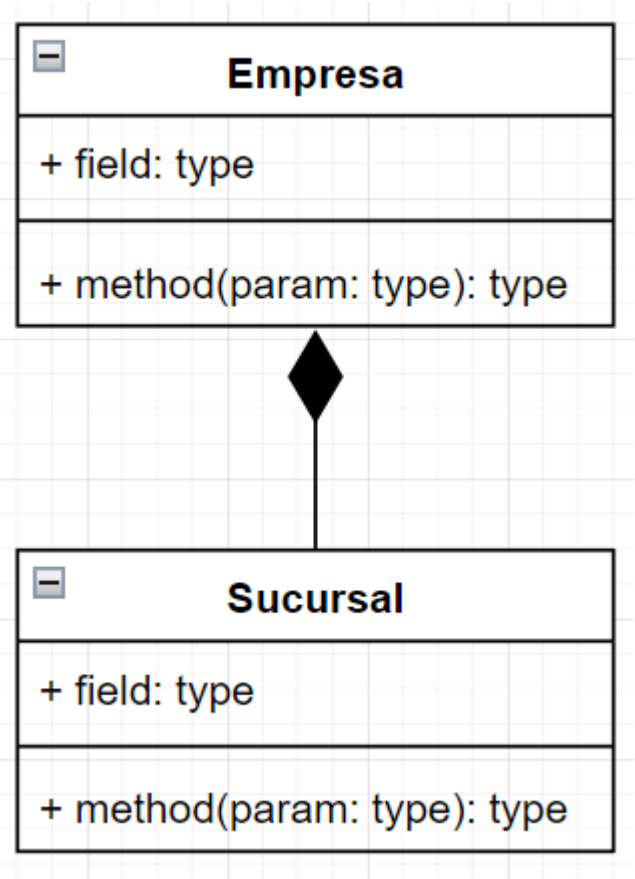

OOO! AN ARTiST!
WOW
I WONDER WHAT HE'S DRAWiNG
iS HE DRAWiNG ...
ME??
M.PATRINOS/BUZZFEED
Document
-authors : String[ ]
-date : Date
+getAuthors() : String[ ]
+addAuthor(name) : void
+getDate() : date
Book
-title : String
+getTitle() : String
EMail
-subject : String
-to : String[ ]
+getSubject() : String
+getTo() : String[ ]

Diagrama de clases UML - Relaciones de composición

La composición es similar a la agregación, representa una **relación jerárquica entre un objeto y las partes que lo componen, pero de una forma más fuerte**. En este caso, **los elementos que forman parte no tienen sentido de existencia cuando el primero no existe**. Es decir, cuando el elemento que contiene los otros desaparece, deben desaparecer todos ya que no tienen sentido por sí mismos sino que dependen del elemento que componen. Además, suelen tener los mismos tiempo de vida. Los componentes no se comparten entre varios elementos, esta es otra de las diferencias con la agregación.

Se representa con una línea continua con un rombo relleno en la clase que es compuesta.

Un ejemplo de esta relación sería: «Una empresa se compone de sucursales»



Relaciones de composición y agregación

La diferencia entre agregación y composición es semántica, por lo que a veces no está del todo definida. Ninguna de las dos tienen análogos en muchos lenguajes de programación.

Un «agregado» representa **un todo que comprende varias partes**; de esta manera, un Comité es un agregado de sus Miembros. Una reunión es un agregado de una agenda, una sala y los asistentes. En el momento de la implementación, esta relación no es de contención. (Una reunión no contiene una sala). Del mismo modo, las partes del agregado podrían estar haciendo otras cosas en otras partes del programa, por lo que podrían ser referenciadas por varios objetos que nada tienen que ver. En otras palabras, no existe una diferencia de nivel de implementación entre la agregación y una simple relación de «usos». En ambos casos, un objeto tiene referencias a otros objetos. Aunque no existe una diferencia en la implementación, definitivamente vale la pena capturar la relación en el diagrama UML, tanto porque ayuda a comprender mejor el modelo de dominio, como porque puede haber problemas de implementación que pueden pasar desapercibidos. Podría permitir relaciones de acoplamiento más estrictas en una agregación de lo que haría con un simple «uso», por ejemplo.

La composición, por otro lado, implica **un acoplamiento aún más estricto que la agregación**, y definitivamente implica la contención. El requisito básico es que, **si una clase de objetos (llamado «contenedor») se compone de otros objetos (llamados «elementos»), entonces los elementos aparecerán y también serán destruidos como un efecto secundario de crear o destruir el contenedor**. Un ejemplo podría ser el nombre y la dirección del Cliente. Un cliente sin nombre o dirección no tiene valor. Por la misma razón, cuando se destruye al cliente, no tiene sentido mantener el nombre y la dirección. (Compare esta situación con la agregación, donde destruir al Comité no debe causar la destrucción de los miembros, ya que pueden ser miembros de otros Comités).

Diagrama de clases UML - Relación de dependencia

Se utiliza este tipo de relación para representar que **una clase requiere de una funcionalidad de otra para ofrecer sus servicios**. Es una relación más débil que la asociación y se representa con una **flecha discontinua** que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma, es decir, **de la clase cliente a la clase proveedora**.

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”.

Diagrama de clases UML - Relaciones de Herencia

Otra relación muy común en el diagrama de clases es la herencia. Este tipo de relaciones permiten que una clase (clase hija o subclase) reciba los atributos y métodos de otra clase (clase padre o superclase). Estos atributos y métodos recibidos se suman a los que la clase tiene por sí misma. Se utiliza en relaciones «es un».

Un ejemplo de esta relación podría ser la siguiente: Un pez, un perro y un gato son animales.

(Veremos herencia más adelante)

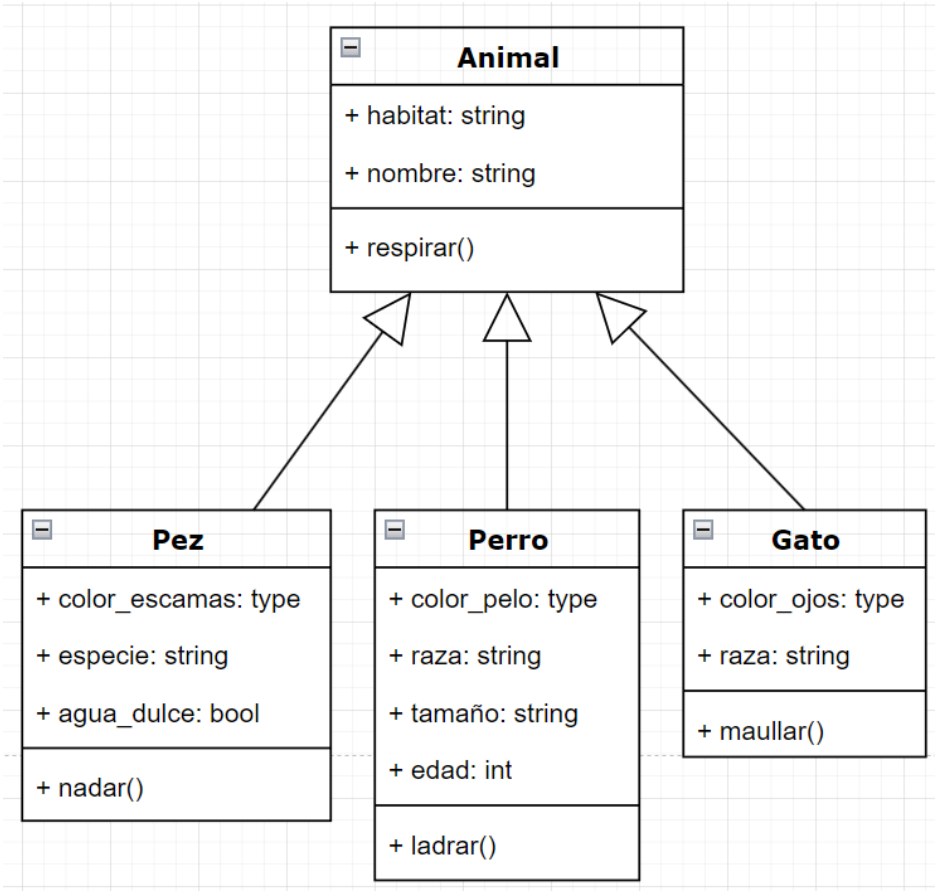


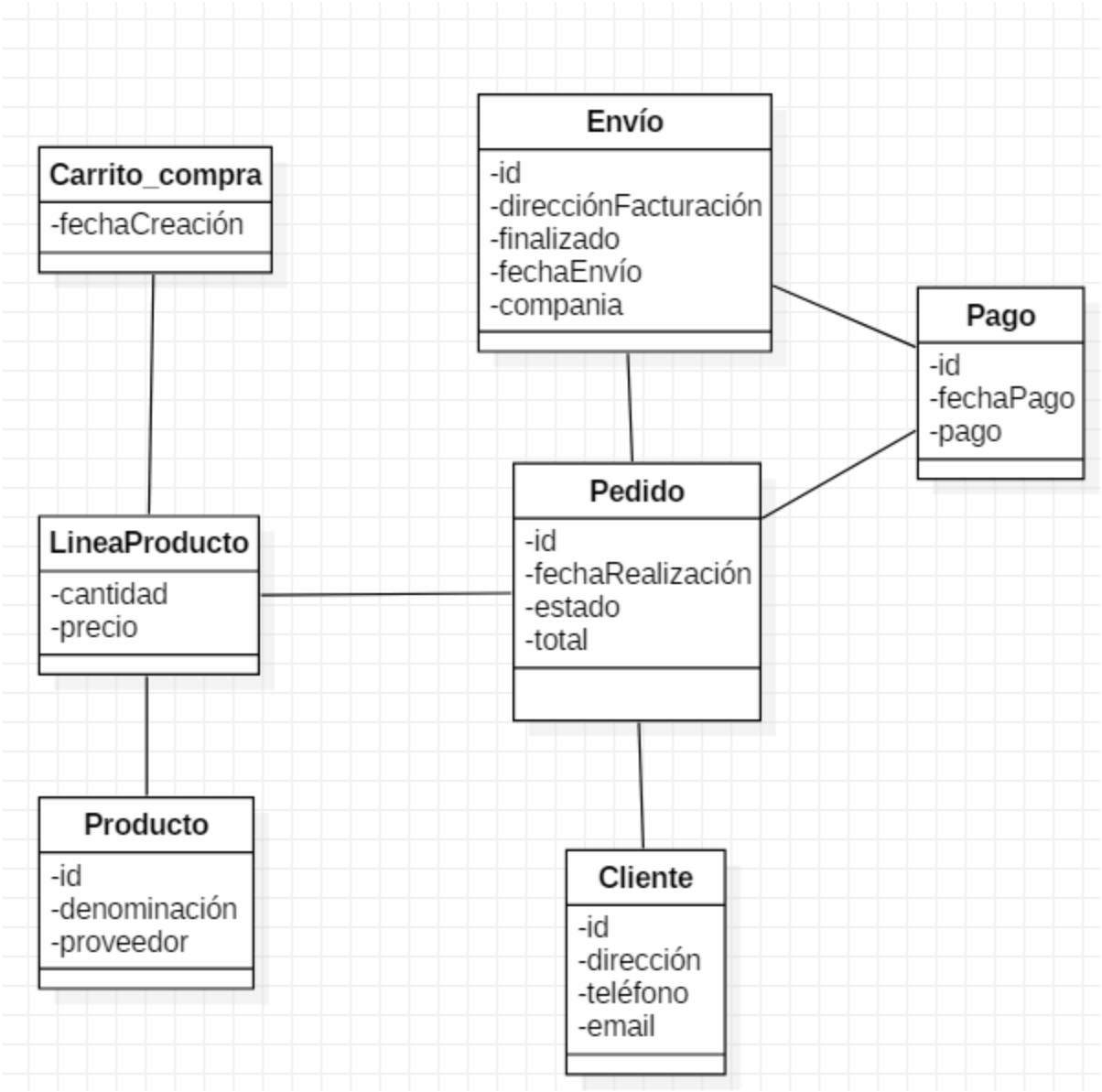
Diagrama de clases UML - Multiplicidad de las relaciones

Multiplicidad	Significado
1	Sólo uno
0..1	Cero o Uno
N..M	Desde N hasta M incluidos, con 0 < N < M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)
n	Un número *n* determinado

Observaciones:

Cuando la multiplicidad mínima es 0, la relación es opcional.

Cuando la multiplicidad es de al menos 1, la relación es obligatoria.


Carrito_compra
-fechaCreación
Envío
-id
-direcciónFacturación
-finalizado
-fechaEnvío
-compania
Pago
-id
-fechaPago
-pago
LineaProducto
-cantidad
-precio
Pedido
-id
-fechaRealización
-estado
-total
Producto
-id
-denominación
-proveedor
Cliente
-id
-dirección
-teléfono
-email

How the UML diagram describes the software

How the code is actually written

Recordando: Importancia de la documentación

Como vimos anteriormente, el resultado de la etapa de Diseño es un **documento** elaborado por uno o más diseñadores, que describen los módulos que integrarán el sistema y la forma en que se relacionan entre sí. El documento también puede establecer casos de prueba o tests que se aplicarán en la etapa de verificación.