Conceptos previos

POO: La POO es una forma de escribir programas que se basa en imitar el mundo real. En vez de solo usar funciones y datos por separado, en la POO usamos objetos que combinan ambos.

Por ejemplo, estos bloques, podríamos abstraer la idea a **POO**, en la que la **clase** “bloque”, tiene una serie de **Atributos,** que si bien, no hace que sean todos iguales, los hace parecidos, es decir, todos tienen el mismo tamaño y 6 caras, pero en esencia son distintos, para hacer mas clara la idea está lo siguiente:

```
class Bloque:

    def __init__(self, id, name, dimensions, texture):
        this.id = id
        this.name = name
        this.dimensions = dimensions
        this.texture = texture
```

Esta sería nuestra clase bloque, esto nos supone una “plantilla”, que puede usarse la cantidad de veces que sea necesario en nuestro programa para crear instancias.

Instancia: una instancia es un objeto específico que se crea a partir de una clase, continuando con el ejemplo, cada que se quiera crear un bloque nuevo, no necesitamos crear una clase entera para crear un nuevo “material”, en su lugar creamos una instancia.

```
bloque_piedra = Bloque(
  id=1,
  name="Piedra",
  dimensions=(1, 1, 1),
  texture="piedra.png"
)

bloque_hierba = Bloque(
  id=2,
  name="Hierba",
  dimensions=(1, 1, 1),
  texture="hierba.png"
)
```

He aquí, tenemos 2 bloques, a partir de la misma clase, con características similares, en parámetros distintos.

Encapsulamiento: Esto se refiere al modificador de acceso de métodos y variables.

- Métodos
  - def funcion(self): se usa para público, cualquiera puede acceder al atributo o método
  - def __funcion(self): representa privado, sólo la propia clase puede acceder
  - def _funcion(self): representa protegido, accesible por la clase y sus subclases
  - @staticmethod def funcion(): Estático, no necesita self, no accede a la instancia. Pertenece a la clase.
- Variables
  - Público, variable

- Protegido, _variable
- Privado, __variable
- Estático, basta con dejarla fuera del __init__
- Constante, no existe de forma directa en python, por convención sólo se deja en mayúsculas

Plantilla básica elicitación de requisitos

Identificación del requerimiento:	RF01 El identificador de requisitos va en orden numérico ascendente, con la estructura mostrada, RFXXX, esto es importante ya que será usado en el diagrama de casos de uso para facilitar la comprensión del mismo.
Nombre del Requerimiento:	Este campo no tiene mayor dificultad, simplemente se le da un nombre que describa su función
Características:	Esta sección es utilizada para describir brevemente al requisito planteado, a fin de dar una idea general de lo que hace.
Descripción del requerimiento:	Para esta parte, se realiza una descripción más detallada del requisito planteado, incluyendo especificaciones sobre el proceso que llevará a cabo.
Requerimiento NO funcional:	• RNF03 Esta sección especifica si existe algún requisito no funcional que dependa o interactúe directamente con el requisito planteado, pueden ser uno o más, incluso ninguno, para ese caso particular se llena con un "N/A"
Prioridad del requerimiento: Para esta parte, se debe evaluar por alguno de los métodos de priorización que deseen, este rango varía generalmente en 3 clasificaciones, ALTA, MEDIA, BAJA, en mayúsculas.	

<Los requisitos no funcionales tienen el mismo formato, omitiendo la casilla de Requerimiento NO funcional>

Ahora bien, ¿cómo distinguir un requisito funcional de uno no funcional?

- **RF:** estos describen funciones y características que el sistema DEBE llevar a cabo, enfocándose en lo que el sistema hará y cómo lo hará
- **RNF:** describe las características y propiedades, más específicamente acerca del comportamiento y los estándares de calidad, como el rendimiento, seguridad, usabilidad, componentes de hardware, mantenibilidad, etc

Cabe resaltar que es necesario que los requerimientos sean

- **ÚNICOS**
- **NO SEAN AMBIGUOS**
- **PUEDAN VERIFICARSE**
- **IMPLEMENTABLES**

Métodos de selección de prioridad

Pues bien, para esta sección hay varias opciones, voy a listar las más conocidas.

Método MoSCoW

Requisito	Tipo	Prioridad (MoSCoW)	Valor Usuario	Complejidad	Comentario
Descripción breve del requisito en un lenguaje entendible para usuarios	especificar si es funcional o no funcional	**Must (Debe tener):** Indica que es necesario para el funcionamiento básico del sistema. Si no se implementa, el sistema no cumple su objetivo. **Should (Debería tener):** Importante, pero no esencial, el sistema puede mantenerse en su ausencia.. **Could (Podría tener):** Deseable, pero no necesario. Se implementa si hay tiempo o recursos extra. **Won't (No por ahora):** Se reconoce como útil, pero está fuera del alcance actual del proyecto. Puede considerarse para versiones futuras.	Que tanto valor le dará el usuario final a la funcionalidad, este varía entre ALTO/ MEDIO /BAJO o puede dársele un puntaje numérico	Dificultad esperada por parte del equipo de desarrollo e igualmente se califica entre ALTO/MEDIO/ BAJO o con puntaje numérico, generalmente de 1 a 5	Razones del porqué es necesario,riesgos si no se implementa, dependencias con otros requisitos y normativas asociadas si aplica

"Buy a Feature" es una técnica de priorización en la que se simula que cada participante dispone de un presupuesto limitado (por ejemplo, $100) y debe “comprar” las funcionalidades o requisitos que considera más valiosos para el proyecto.

Para aplicar esta técnica, primero se debe listar todos los requisitos y asignarles un precio estimado en función del esfuerzo necesario para implementarlos. Este valor puede calcularse de manera relativa, tomando como referencia la complejidad técnica o la cantidad de horas de trabajo requeridas. Una vez definidos los precios, se establece el presupuesto máximo que podrá gastar cada participante.

Luego, a través de una encuesta o dinámica grupal, se consulta a los involucrados en el proyecto como stakeholders, usuarios o desarrolladores. cuáles funcionalidades “comprarían” con su presupuesto asignado. Esto permite obtener una perspectiva colectiva sobre cuáles requisitos son percibidos como más valiosos o prioritarios, facilitando así la toma de decisiones en el desarrollo.

Requisito	**Precio**	**Comprado**
Nombre o identificador del requisito	$-	SI/ N/A

Posteriormente se tabulan los resultados de la siguiente forma:

Requisito	**Total Recaudado**
Nombre o identificador del requisito	$-

En cuanto a los diagramas, hay muchas herramientas libres o de paga, pero personalmente suelo usar draw.io es comodo y permite la colaboración de equipo y siento que otorga más libertad

Diagrama de Casos de uso

Este diagrama debería ser el más simple de la lista si se tiene un buen conocimiento de los requisitos del sistema.

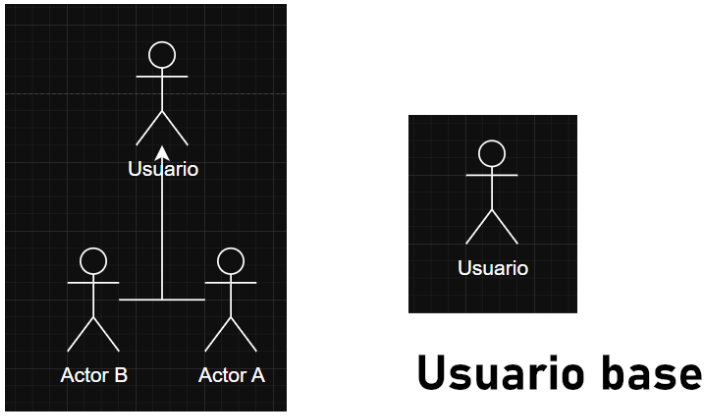
Diagrama de casos de uso general:

- Este diagrama ilustra a cada uno de los requisitos y cómo interactúan entre ellos y con el usuario, se realiza **DESPUÉS DE LOS DIAGRAMAS DE CASO DE USO INDIVIDUAL**

Por cada caso de uso identificado y utilizado para el diagrama de caso de uso general, se debe llenar una tabla así, a fin de especificar detalladamente las vías de acción de cada uno de los casos de uso.

Nombre	**CU01:**
Descripción corta	
Precondición	Condiciones que se deben tener en cuenta antes de interactuar con el caso de uso
Postcondición	Resultados esperados tras la operación o intervención del caso de uso a documentar
Situaciones de error	Situaciones que puedan generar errores que puedan ser previsibles con antelación
Estado del sistema en el caso de error	Especificar qué sucederá con el sistema en caso de que ocurra algún error
Actor	Nombre de la entidad que interactúa con el sistema en un caso de uso dado, como un servidor, un usuario, clientes, administrativos, recepcionistas, etc
Desencadenante	Qué acción hace que el caso de uso entre en acción
Proceso estándar	Descripción detallada del proceso esperado por el sistema ante la interacción del actor
Proceso alternativo	Descripción detallada del proceso que ocurre en caso de que se presente el error

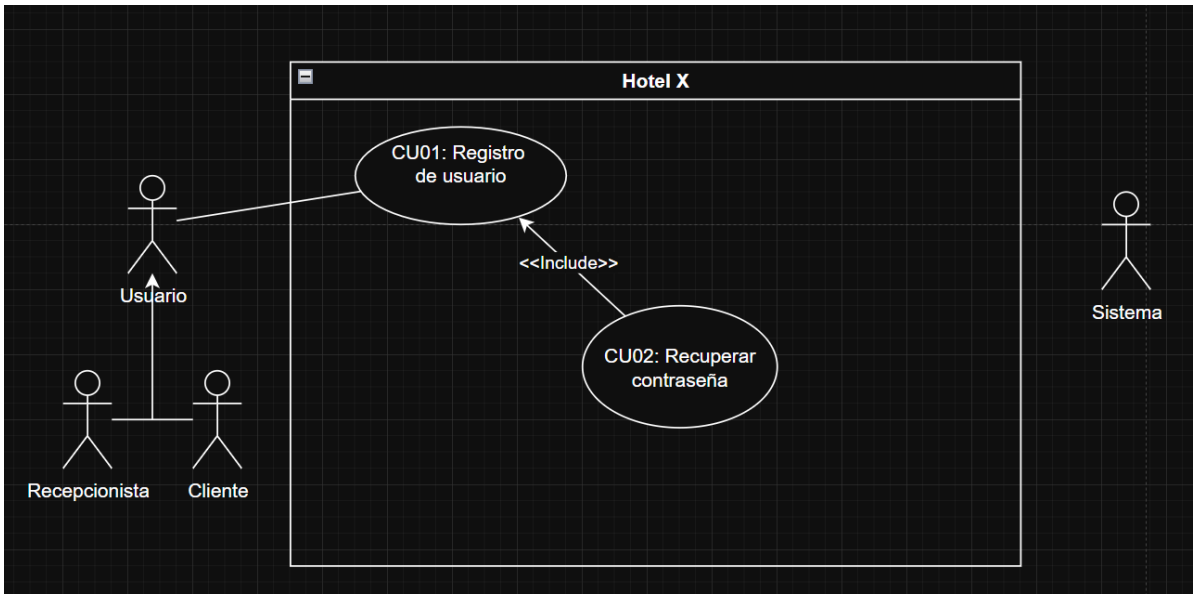
- **Notación de actores del sistema**



Figura[1]: Notación de usuario Diagrama de casos de uso

Normalmente solo se usaría la representación del “usuario base”, pero cuando varios actores distintos interactúan con el sistema y se convierten en otro tipo de actor, se aplica la generalización, que es juntar el trazo de los N actores con sus nombres al nombre del actor general.

- **Sistema y componentes**



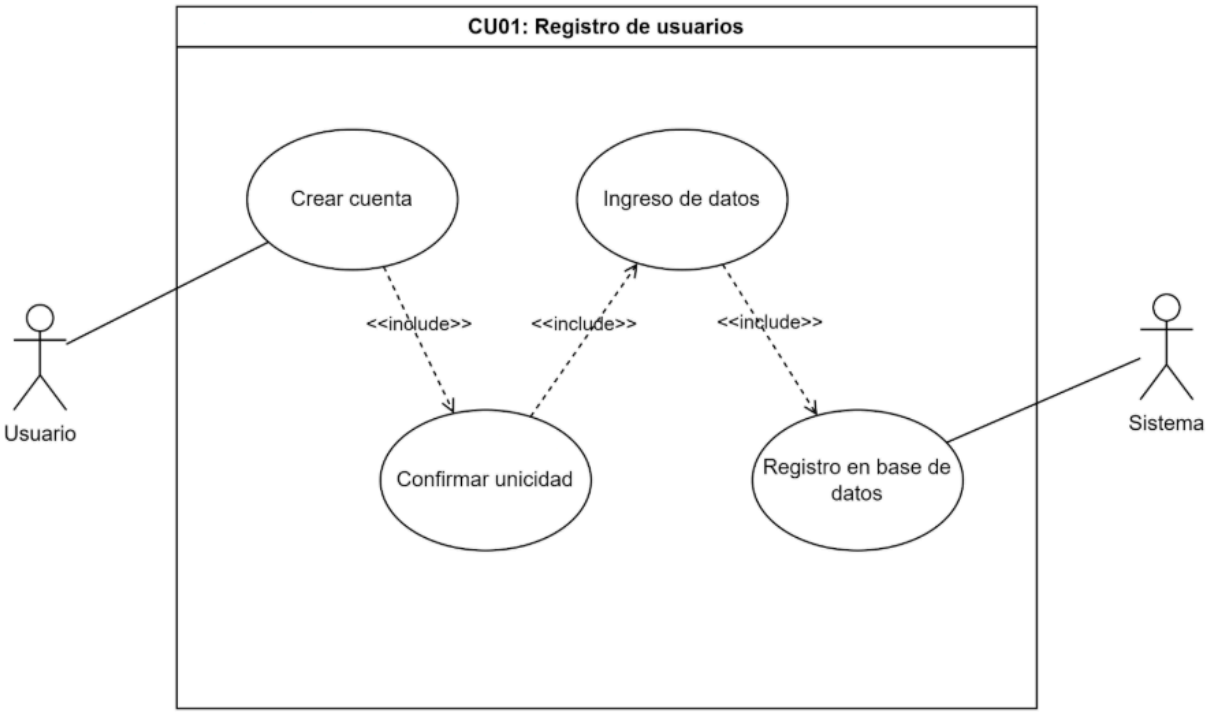
Figura[2]: ejemplo de diagrama de caso de uso general

- Es importante que cada uno de los óvalos que representan casos de uso esté dentro del sistema, representado por un recuadro, con su respectivo título.

Aquí se presenta un detalle nuevo, el include y el extend, ¿Qué son?

- Include significa que el caso de uso del que parte la flecha, requiere la existencia del que está en la punta.
- Extend se usa cuando la intervención del segundo caso de uso es opcional

Diagramas de caso de uso individual:



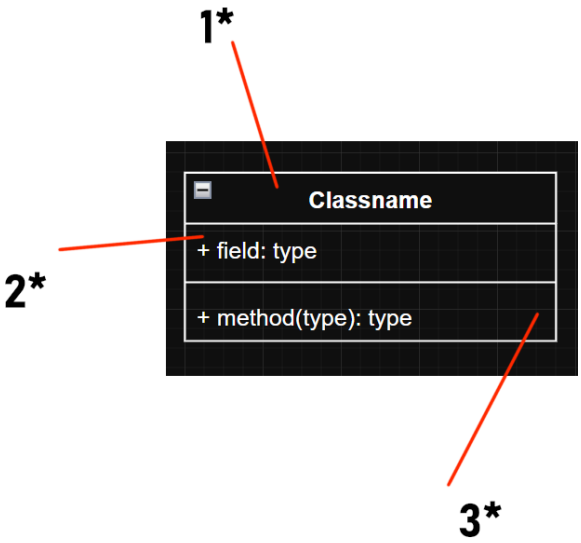
Figura[3]: ejemplo de diagrama de caso de uso individual

<Nota del autor, las relaciones entre actor y caso de uso son líneas rectas, **NO FLECHAS**
Adicionalmente se le pueden dar colores al diagrama para hacerlo más legible, ya es a su elección>

Diagrama de clases

Este tipo de diagrama posee una serie de elementos que se verán a continuación:

- **Clases**
- **Atributos**
- **métodos**
- **Relaciones**

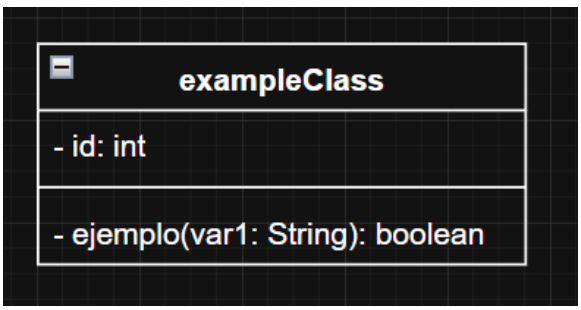


Figura[4]: Representación de una clase.

- **1* Es el campo para el nombre de la clase en función de las convenciones del lenguaje de programación a usar, como camelcase u otras alternativas.**
- **<u>2* Este espacio está destinado para las variables propias de la clase junto a sus modificadores de acceso</u>**

- **+** se usa para público, cualquiera puede acceder al atributo o método
- - representa privado, sólo la propia clase puede acceder
- # representa protegido, accesible por la clase y sus subclases
- ~ representa accesible solo para el mismo paquete, es el estado por defecto
- Estático o constante, solo se subraya

- **3* Este último campo es para añadir los métodos/funciones de la clase, usando los mismo modificadores de acceso, pero este campo tiene un par de detalles más.**
  - **la sintaxis se compone de (modificador de acceso) nombre del metodo(parametros que recibe: tipo de dato del parámetro): tipo de retorno**
  - **Los métodos constructores no se especifican obligatoriamente en este diagrama**
  - **En caso de que se tenga algun metodo que no retorna nada se deja como :void**



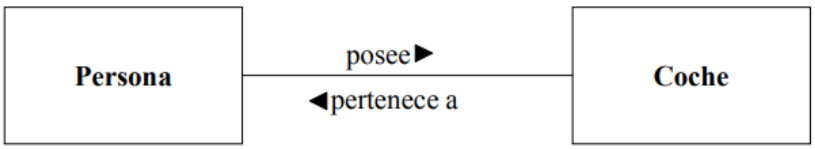
Figura[5]: Ejemplo de clase

<u><Nota del autor, no olviden las buenas prácticas para nombrar métodos clases y variables propias de cada lenguaje de programación></u>

Ahora que se sabe como crear una clase, es hora de ver cómo relacionarlas entre sí, la notación UML tiene una serie de componentes para ello, haciendo afín al concepto de POO.

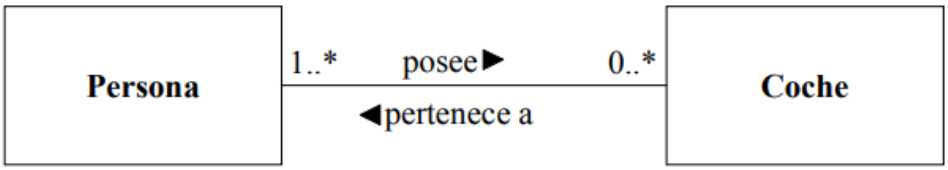
ASOCIACIÓN:

- especifica que los objetos de una clase están conectados con los objetos de otra, esta relación tiene características específicas, como lo son:
    - Nombre
        - La relación entre 2 entidades debe tener un nombre según su cardinalidad



Figura[6]: Asociación simple

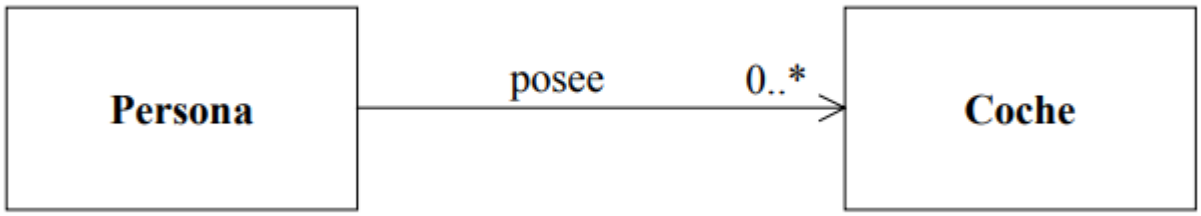
- Multiplicidad



Figura[7]: Ejemplo de multiplicidad.

- Este parámetro especifica la cantidad de objetos que entran en juego en la relación
  - Adicionalmente cuenta con una notación propia, compuesta por números, puntos y "*" y un orden.
  - leyendo de izquierda a derecha, en este caso representa que 1 o varias personas puede poseer, múltiples coches o ninguno. El "*" representa una cantidad no definida de una X cosa.

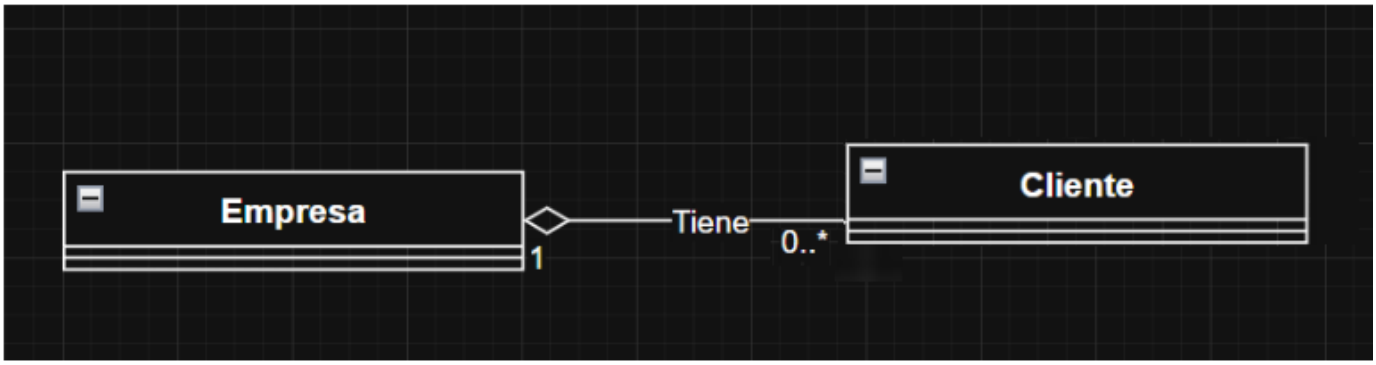
- Restricciones
  - Para este caso, está la restricción de que únicamente una persona puede tener o 0 o muchos coches y la relación es unidireccional por parte de persona, por eso solo hay una flecha de una única punta



Figura[8]: Ejemplo de restricción en una relación entre clases

AGREGACIÓN:

La agregación indica que una clase es parte de otra(composición débil). Esto permite que los componentes puedan ser compartidos por varias clases y la eliminación de una clase no conlleva a la destrucción de la otra, se representa así:

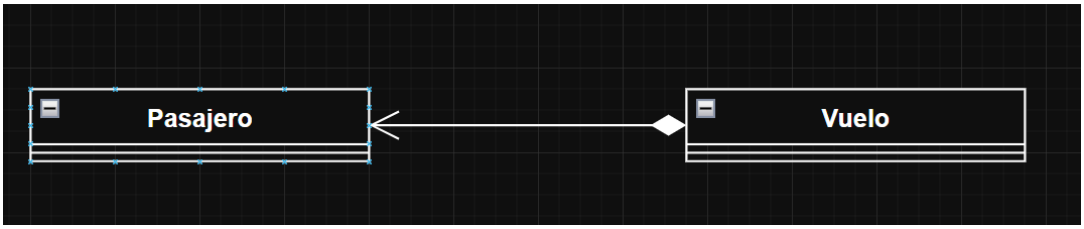


Figura[9]: Ejemplo de agregación

Una empresa puede tener 0 o muchos clientes, sin embargo, la empresa existe incluso si no tiene clientes.
<Esto es conocido como “Cardinalidad”>

COMPOSICIÓN

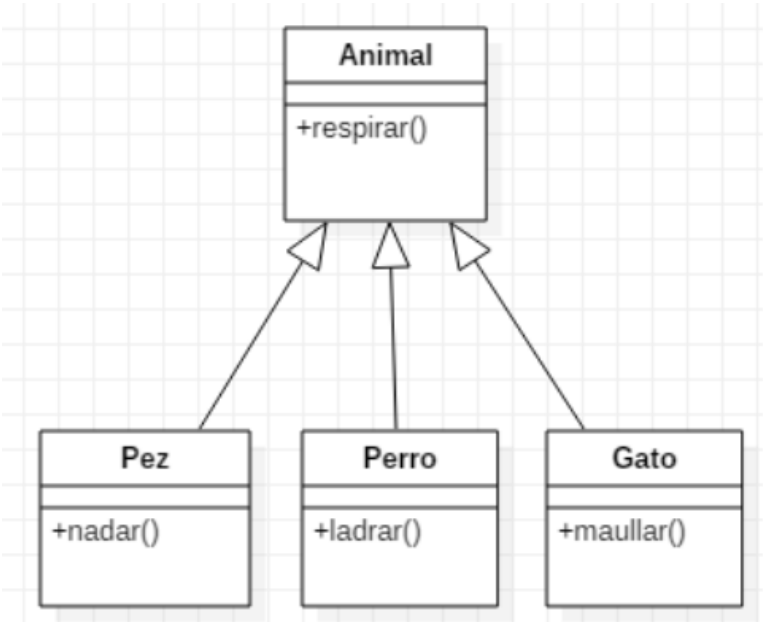
Esta relación hace que si alguna de las clases involucradas desaparece, inmediatamente la otra queda inutilizable



Figura[10]: Ejemplo de composición [se recomienda evitarlo por cuestiones de acoplamiento]

HERENCIA

Generalmente se usa para que una subclase o clase “hija” reciba atributos y métodos de otra clase, sumándose a los propios de la clase



Figura[11]: Ejemplo de herencia

DEPENDENCIA

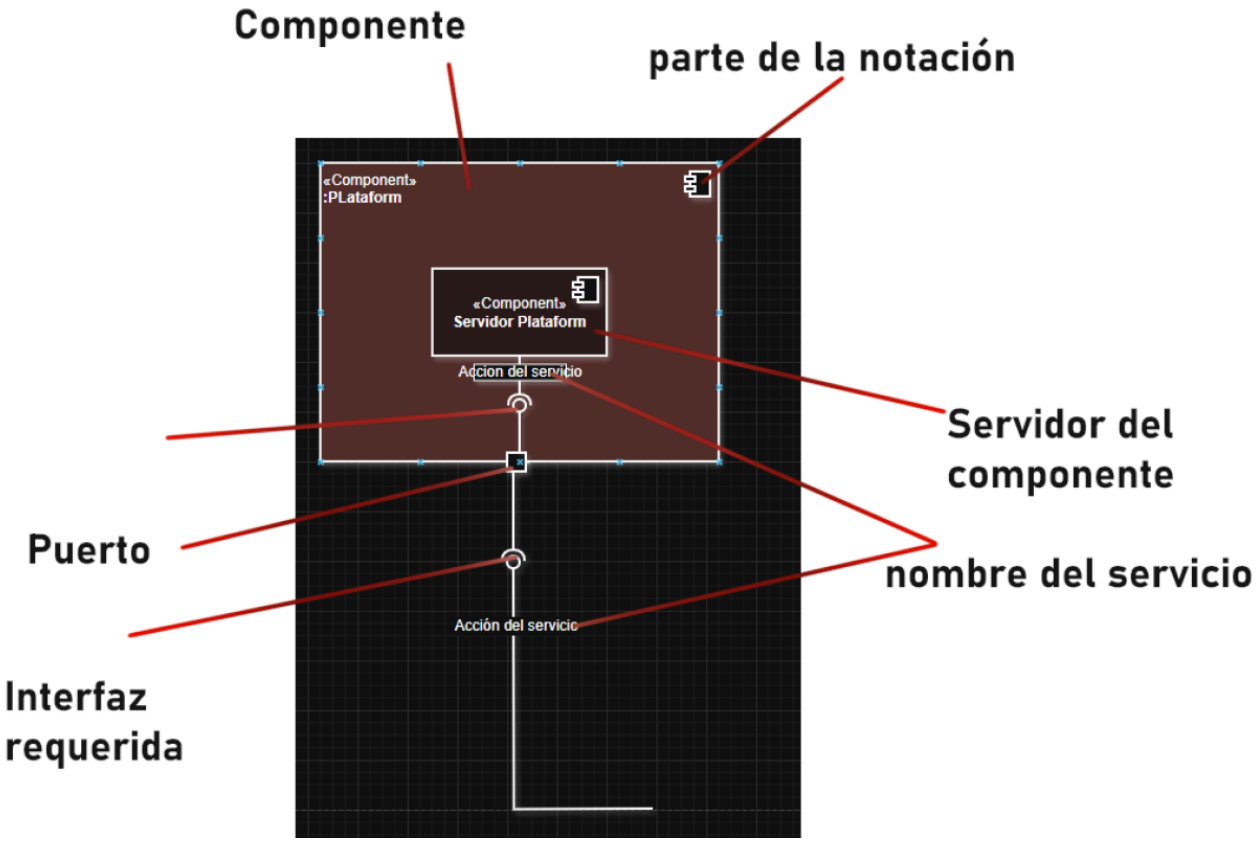
Una dependencia en un diagrama de clases es una relación débil y temporal entre dos clases, donde una necesita usar a la otra para realizar alguna operación, pero no la contiene como atributo permanente, este se representa con una flecha de línea punteada y con la punta sin relleno.

Diagrama de componentes

Este diagrama ilustra cómo se divide un sistema en componentes modulares y cómo se conectan entre sí, en donde un componente cumple una función específica.

La notación presentada está dada para las siguientes entidades:

- **Componente:** Parte de un sistema que tiene su propia funcionalidad y comportamiento.
- **Interfaz:** Una interfaz es un método o conjunto de métodos utilizados por otro componente.
- **Puerto:** Un lugar donde un componente interactúa con su entorno y envía mensajes.
- **Conector:** Una forma en que dos componentes pueden comunicarse.



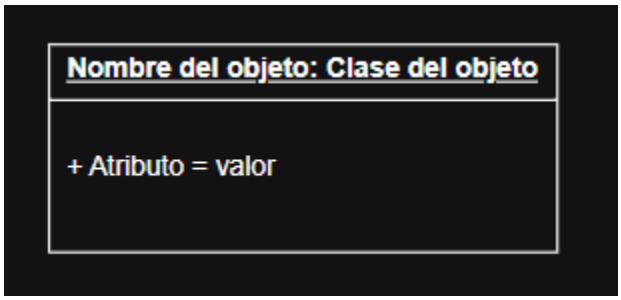
Figura[12]: Ejemplo de diagrama de componentes.

- Un componente puede anidar a otros componentes, si solo anida uno es innecesario y lo mejor es poner el componente general.
- El medio círculo son las interfaces requeridas, el círculo completo son las interfaces ofrecidas.
- Los puertos, se debe añadir uno por cada interfaz que ofrezca o solicite el componente.
- Se le pueden añadir colores para que sea más entendible.

Dado que es un poco más complejo, adjuntaré un ejemplo:
https://drive.google.com/file/d/1-PQOvks3r35Oc-H0_yiMIVGfT51PWZ-8/view?usp=sharing
<No editar>

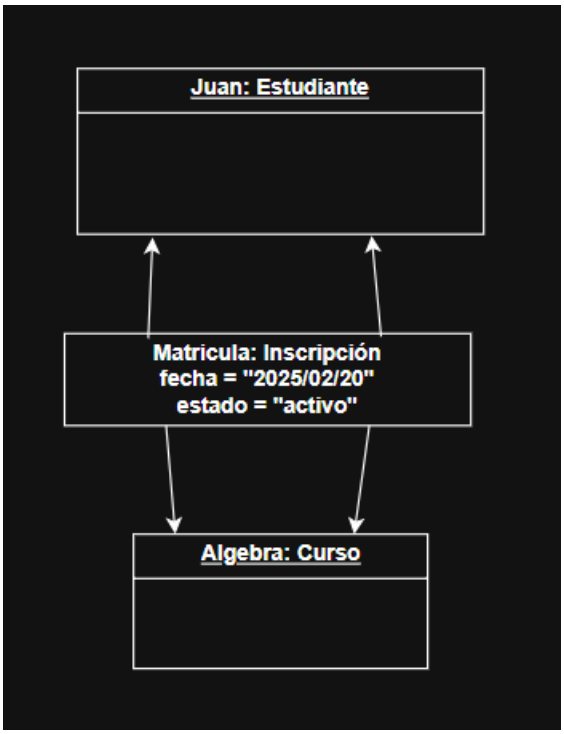
Diagrama de objetos

Si bien la notación es similar a la del diagrama de clases, no necesariamente representan lo mismo, debido a que se utilizan para modelar los elementos que están presentes en un diagrama de clases en un momento determinado.



Figura[13]: Ejemplo de diagrama de objetos.

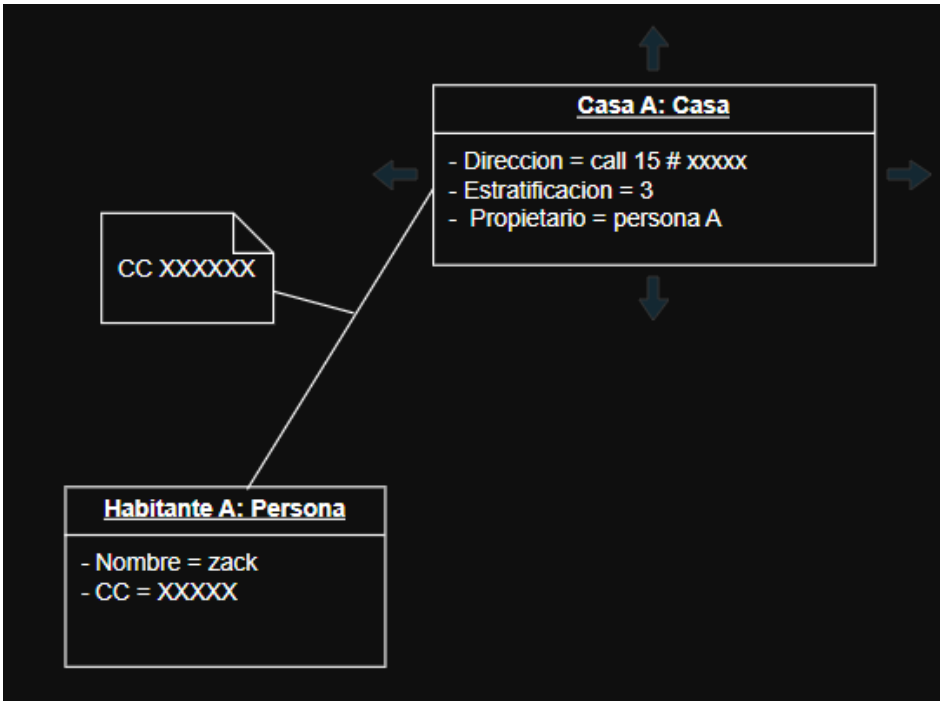
- En este diagrama lo que se busca representar son instancias de objetos, a continuación un ejemplo.
- Los nombres de las relaciones son diferentes también, teniendo entre ellas: l
    - **Link:** Es una relación entre dos objetos que interactúan entre sí.
    - **Directed link:** Es una asociación unidireccional entre dos objetos, es decir, solo uno de los objetos conoce al otro. Se representa con una flecha de línea sólida que termina en una punta rellena
    - **Link object:** Es un objeto que representa una instancia de una asociación entre dos objetos. Se utiliza cuando la asociación tiene atributos propios.



Figura[14]: Ejemplo de Link Object

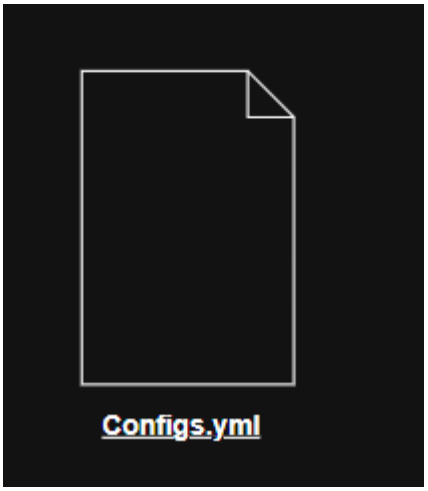
Aqui Matrícula es un link object que representa la relación entre el estudiante Juan y Álgebra, junto a la información que pasó de una clase a otra

- Cabe resaltar que entre relaciones si se debe especificar algo, como la información que se transmite de una clase a otra se usan las “anotaciones”, representadas con:



Figura[15]: Ejemplo anotación

- Artefactos, son objetos “concretos”, como, archivos de código fuente, ejecutables, scripts de configuración, manuales, etc
- Se representan con el mismo símbolo que las anotaciones



Figura[16]: Ejemplo de Artefacto

- Componentes, son unidades de comportamiento, como, servicios web, bibliotecas, módulos de aplicaciones, etc.

Figura[17]: Ejemplo de Componente

Este al ser un módulo, puede tener varias clases dentro u otros elementos vistos anteriormente según sea necesario

PROYECTOS EJEMPLO:
https://docs.google.com/document/d/1zxJA9eRZpE9iR9tJSrQXxRd-PVp1gvENkIhSpT5Un1A/edit?tab=t.0#heading=h.gjdgxs
Requisitos

https://docs.google.com/document/d/1gBIjtDSUHB5P35GPSab92yk59uXAaacNTxxLlf77fpw/edit?tab=t.0
Casos de uso

Referencias:

https://topicosbasededatos997954789.wordpress.com/wp-content/uploads/2019/06/copia-de-ejemplo-formato-ieee-830.pdf
diagramasuml.com/componentes/
https://people.eecs.ku.edu/~hossein/810/Readings/UML-diagrams.pdf
http://www.vc.ehu.es/jiwotvim/IngenieriaSoftware/Teoria/BloqueII/UML-3.pdf
https://www.omg.org/spec/UML/2.5.1/PDF
https://creately.com/blog/es/diagramas/relaciones-de-diagrama-de-clases-uml-explicadas-con-ejemplos/