EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: DISEÑO DE DIAGRAMA DE CLASES PARTIENDO DEL PARADIGMA ORIENTADO A OBJETOS.

EXERCISE 1: DISEÑO DE DIAGRAMA DE CLASES PARTIENDO DEL PARADIGMA ORIENTADO A OBJETOS.

La Programación Orientada a Objetos (POO) es un modelo de programación informática que organiza el diseño de software en torno a datos u objetos, en lugar de funciones y lógica. Un objeto se puede definir como un campo de datos que tiene atributos y comportamientos únicos.

Esta se centra en los objetos que los desarrolladores quieren manipular, en lugar de enfocarse en la lógica necesaria para manipularlos (Abstracción). Este enfoque de programación es adecuado para programas que son grandes, complejos, y se actualizan o mantienen activamente.

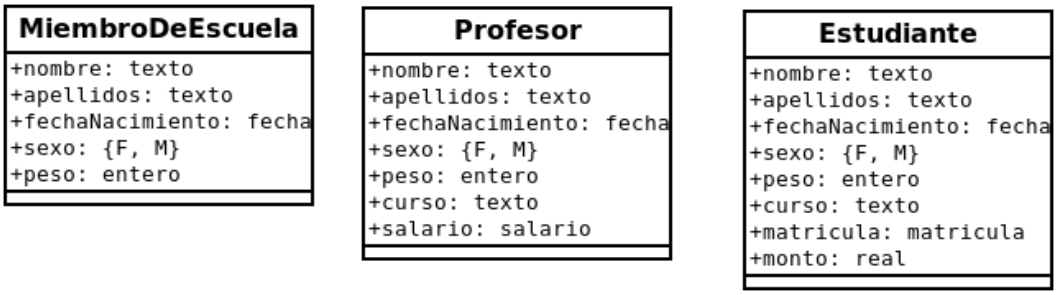
El primer paso en POO es recopilar todos los objetos que un programador desea manipular, e identificar cómo se relacionan entre sí, un ejercicio que a menudo se conoce como modelado de datos.

Los objetos es la manera de organizar datos y de relacionar esos datos con el código apropiado para manejarlo. Son los protagonistas del paradigma de programación llamado Programación Orientada a Objetos. Para modelarlos y crear los diagramas de clases utilizamos la herramienta de software de modelado llamada **Dia.**

Podemos tener un conjunto de objetos que nos represente a personas dentro de una escuela como MiembroDeEscuela, Estudiante y Profesor respectivamente.

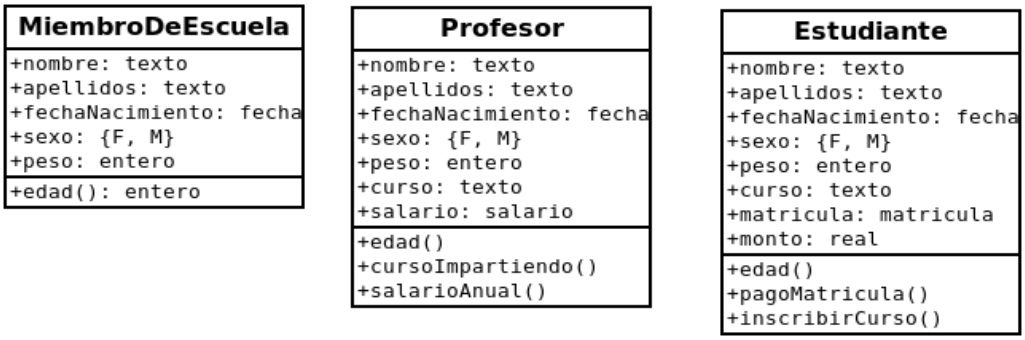
Es importante notar que cuando decimos objetos podemos estar haciendo referencia a dos cosas parecidas, pero distintas.

Por un lado, la definición del tipo, donde se indican cuáles son los atributos y métodos que van a tener todas las variables que sean de ese tipo. Esta definición se llama específicamente, la clase del objeto.



A partir de una clase es posible crear distintas variables que son de ese tipo. A las variables que son de una clase en particular, se las llama instancia de esa clase.

En los métodos podemos definir funcionalidades asociadas a los objetos, que son implantadas o programadas dentro de las clases. En este caso podemos tener un método que permita obtener la edad, el estudiante puede pagar la matrícula, inscribir curso; como también el profesor puede consultar el salario anual y los cursos que está impartiendo.



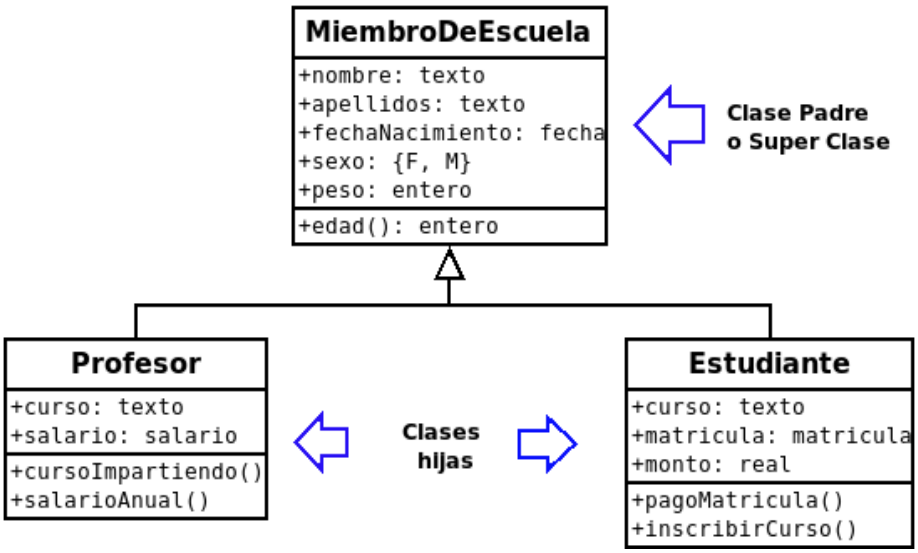
Cada clase puede crear instancias de un objeto, cada uno de ellos con sus atributos definidos de forma independiente, por lo tanto, podemos crear distintas instancias para profesor y estudiantes.

La encapsulación es la implementación y el estado de cada objeto que se mantienen de forma privada dentro de un límite definido o clase. Otros objetos no tienen acceso a esta clase o la autoridad para realizar cambios; por ejemplo, un estudiante no puede modificar o consultar el

salario anual de un profesor si este es privado, pero pueden llamar a una lista de funciones o métodos públicos, como la edad. Esta característica de ocultación de datos proporciona una mayor seguridad al diseño del programa, y evita la manipulación de datos no intencionada.

La herencia de clases es donde se puede asignar relaciones a subclases entre objetos, lo que permite a los desarrolladores reutilizar una lógica común sin dejar de mantener una jerarquía única. Esta propiedad de POO obliga a un análisis de datos más completo, reduce el tiempo de desarrollo, y asegura un mayor nivel de precisión.

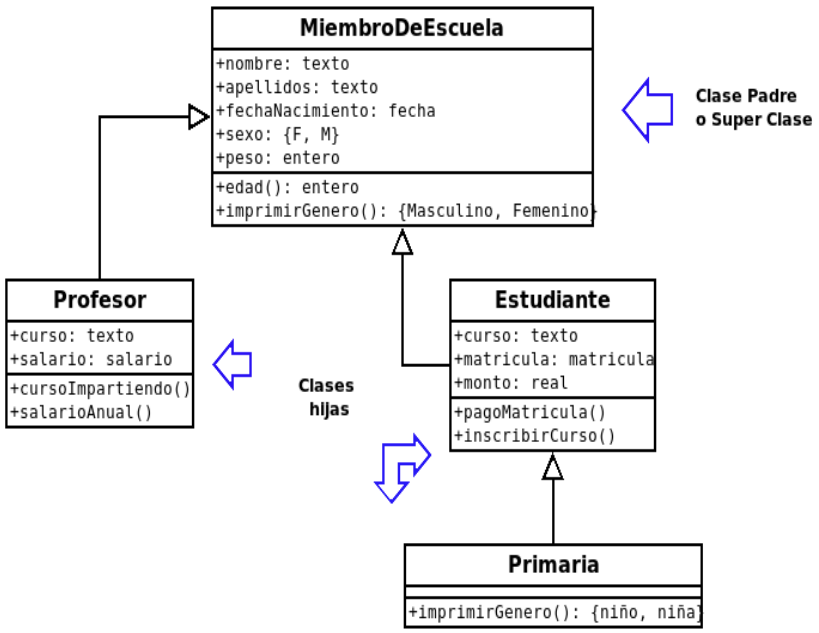
Partiendo de los miembros de la escuela que estamos ejemplificando, contamos con características comunes y métodos que nos da inicio a tener una jerarquía de clases o herencia de clases de los objetos MiembroDeEscuela, Profesor y Estudiante, quedando un diagrama de herencia de la siguiente manera.



Se observa que al aplicar herencia estamos reutilizado el código, ya que tenemos atributos y métodos comunes entre dichos objetos. En este sentido, la herencia consiste en "sacar factor común" de las tres clases (a esta clase se le denomina en la herencia como "Clase Padre o SuperClase") y el método que es específico particularmente en cada clase, se mantiene en la misma, siendo denominadas estas clases como "Clases Hijas", las cuales heredan de la clase padre todos los atributos y métodos públicos o protegidos. Es muy importante decir que las clases hijas no van a heredar nunca los atributos y métodos privados de la clase padre.

El polimorfismo es donde los objetos pueden adoptar más de una forma según su contexto. El programa determinará qué significado o uso es necesario para cada ejecución de ese objeto, reduciendo la necesidad de duplicar código.

Continuando con nuestro ejemplo, podemos agregar un método a la clase MiembroDeEscuela que nos imprima el género masculino o femenino según el sexo (M o F), pero además tenemos estudiantes que, si son de primaria, nos imprima el título según su género como niña o niño respectivamente, nuestro diagrama quedaría de la siguiente manera.



De esta manera observamos el beneficio que nos da el polimorfismo por medio de la herencia. La ventaja del polimorfismo es que cuando necesitamos pasar más subclases, como agregar adolescentes, adultos, entre otros, solo necesitamos heredar el tipo MiembroDeEscuela, y el método imprimirGenero() no se puede invalidar directamente (es decir, usar la de MiembroDeEscuela), también se puede reescribir uno único. Según el tipo de MiembroDeEscuela que se llama, sólo se preocupa por la llamada, independientemente de los detalles, y cuando agregamos una subclase de MiembroDeEscuela, sólo debemos asegurarnos de que el nuevo método esté escrito correctamente, independientemente del código original.