encapsulamientos
son las propiedades de las clases tanto metodos como atributos, sea públicos, privados. y protegidos

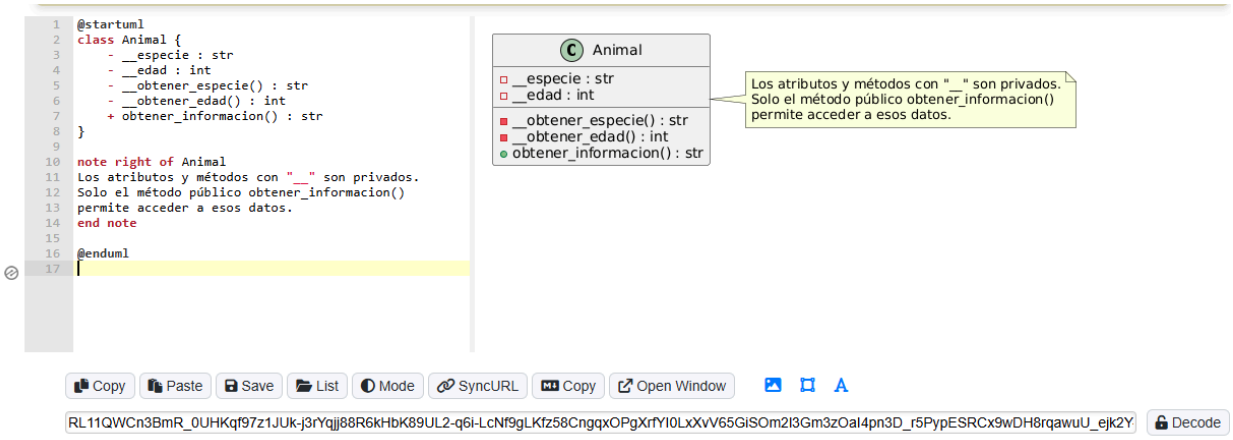
https://meet.google.com/ush-xsin-oaa

ya que vimos

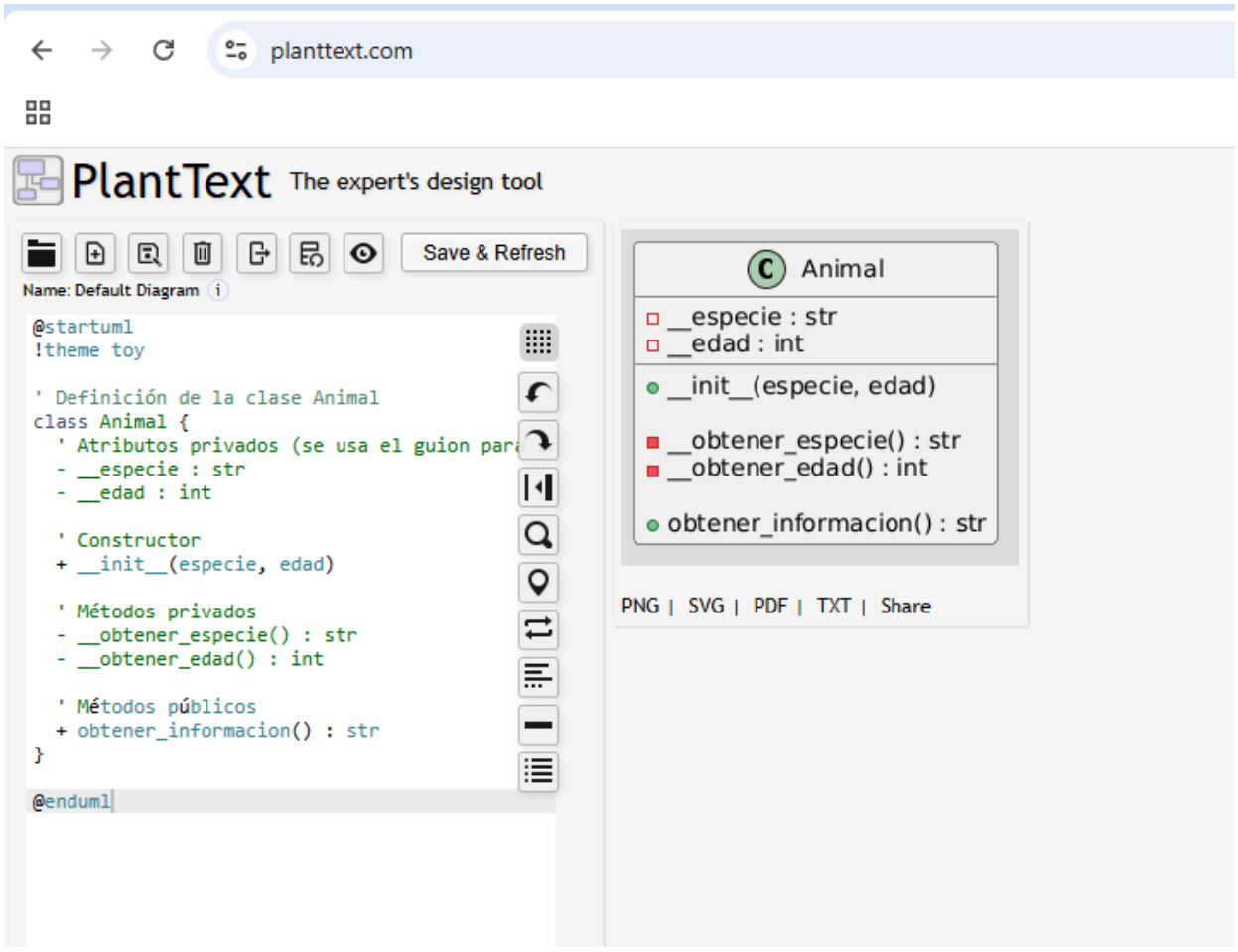
uml
herencia
encapsulamiento publico
instancias, objetos

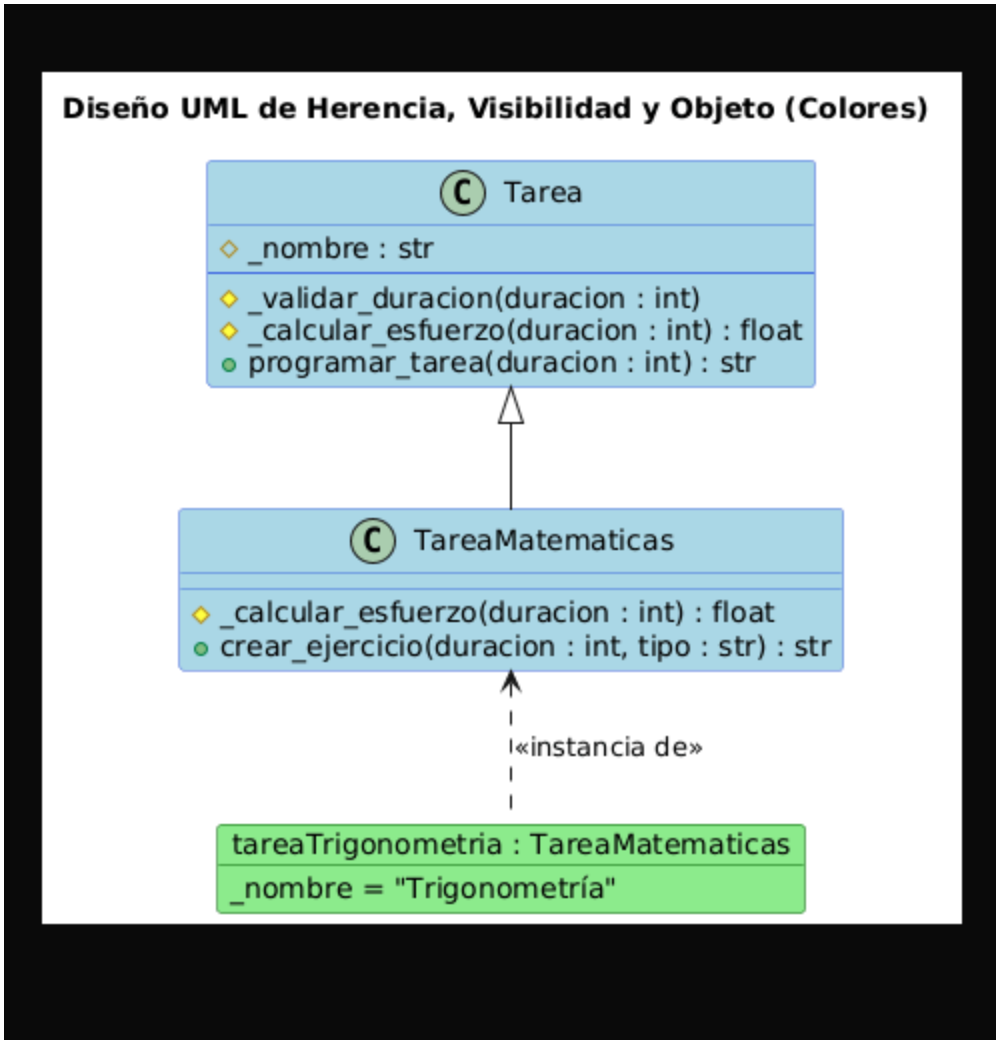
```
# clase animal con metodos y atributos privados

class Animal:
    # Constructor de la clase
    def __init__(self, especie, edad):  # Constructor de la clase
        # atributos privados
        self.__especie = especie  # Atributo privado
        self.__edad = edad  # Atributo privado

   # metodos privados
    def __obtener_especie(self):  # Método privado para obtener la especie
        return self.__especie

    def __obtener_edad(self):  # Método privado para obtener la edad
        return self.__edad
    # Métodos públicos para acceder a los métodos privados
    def obtener_informacion(self):  # Método público para obtener toda la
información
        especie = self.__obtener_especie()
        edad = self.__obtener_edad()
        return f"Especie: {especie}, Edad: {edad}"
# si  la clase tiene metodos y atribustos privados no se puede acceder a
ellos desde fuera de la clase, entonces es necesario que tenga un metodo
publico para acceder a ellos
# objeto de la clase Animal
animal = Animal("Perro", 5)
# Accediendo a los atributos privados a través de métodos públicos
```

```
print(animal.obtener_informacion())   # Salida: Especie: Perro, Edad: 5
```




```
@startuml
!theme toy

' Definición de la clase Animal
class Animal {
  ' Atributos privados (se usa el guion para indicar privado)
  - __especie : str
  - __edad : int

  ' Constructor
  + __init__(especie, edad)

  ' Métodos privados
  - __obtener_especie() : str
  - __obtener_edad() : int

  ' Métodos públicos
  + obtener_informacion() : str
}

@enduml
```


planttext.com
PlantText
The expert's design tool
Save & Refresh
Name: Default Diagram
@startuml
!theme toy
' Definición de la clase Animal
class Animal {
' Atributos privados (se usa el guion para
- __especie : str
- __edad : int
' Constructor
+ __init__(especie, edad)
' Métodos privados
- __obtener_especie() : str
- __obtener_edad() : int
' Métodos públicos
+ obtener_informacion() : str
}
@enduml
Animal
__especie : str
__edad : int
__init__(especie, edad)
__obtener_especie() : str
__obtener_edad() : int
obtener_informacion() : str
PNG | SVG | PDF | TXT | Share



puedes diferenciar las clases lso ojetos con colres y realacionar el objeto

Polimorfismo
clase padres usuario hereda vendedor y administrador , va ser un método mostrar(ejecución diferente)

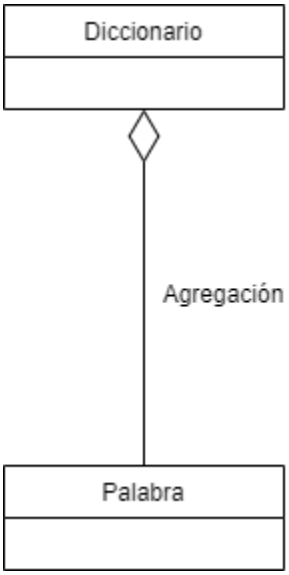
.1Clases	Ok
2. Objetos	Ok
3. Atributos	Ok
4. Métodos	Ok
5. Encapsulamiento	Ok
6. Herencia	Ok

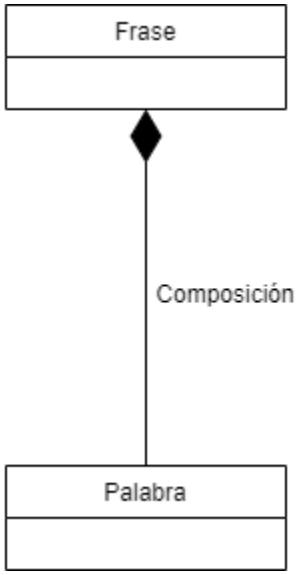
7. Polimorfismo Ok

8. composición ok

9. agregación. ok

10. asociaciones. multiples ok





Relación	**Código Python**	**Notación UML**	**Explicación**
Composición	self.departamentos = [] (lista dentro de Universidad)	**Diamante negro (*--)**	Los departamentos pertenecen *exclusivamente* a una universidad. Su ciclo de vida depende de la universidad.
Agregación	self.profesores = [] (lista dentro de Universidad)	**Diamante blanco (o--)**	Los profesores están *asociados* a la universidad, pero pueden existir de forma independiente

			(tienen su propio ciclo de vida).
Instanciación	profesor1 = Profesor(...)	**Línea punteada (<..)** del objeto a la clase	Muestra que un objeto específico es una instancia de una clase definida.

```
@startuml
' Configuración para la visibilidad de atributos
skinparam classAttributeIconVisible false

' Definición de colores de fondo para las clases usando #27BEF5
skinparam class {
   BackgroundColor #27BEF5
   BorderColor DarkBlue
   FontColor Black
}

' Definición de colores de fondo para los objetos/instancias
skinparam object {
   BackgroundColor PaleGreen
   BorderColor DarkGreen
   FontColor Black
}

title Diseño UML (Composición, Agregación y #27BEF5)

' ========================================================
' CLASES Y RELACIONES
' ========================================================
```

```
class Profesor {
   + nombre : str
   + edad : int
   --
   + mostrar_profesor() : str
}

class Departamento {
   + nombre : str
   + cursos : List<str>
   --
   + agregar_curso(curso : str)
   + mostrar_departamento() : str
}

class Universidad {
   + nombre : str
   + departamentos : List<Departamento>
   + profesores : List<Profesor>
   --
   + agregar_departamento(departamento)
   + agregar_profesor(profesor)
   + mostrar_universidad() : str
}

' ------------------------------------------------------
' RELACIONES ENTRE CLASES
' ------------------------------------------------------

' Composición (Diamante negro)
Universidad "1" *-- "0..*" Departamento : contiene

' Agregación (Diamante blanco)
```

```
Universidad "1" o-- "0..*" Profesor : emplea

' Asociación
Departamento ..> Profesor : trabaja_con

' ============================================================
' OBJETOS/INSTANCIAS
' ============================================================

object "profesor1 : Profesor" as P1 {
   nombre = "Juan"
   edad = 40
}

object "profesor2 : Profesor" as P2 {
   nombre = "María"
   edad = 35
}

object "departamento1 : Departamento" as D1 {
   nombre = "Matemáticas"
   cursos = ["Álgebra", "Cálculo"]
}

object "universidad : Universidad" as U1 {
   nombre = "Universidad Nacional"
}

' ------------------------------------------------------------
' RELACIONES DE INSTANCIACIÓN
' ------------------------------------------------------------
Profesor <.. P1 : <<instancia de>>
Profesor <.. P2 : <<instancia de>>
```

Departamento <.. D1 : <<instancia de>>

Universidad <.. U1 : <<instancia de>>

@enduml