T5_UML (MODELIZACIÓN)

1. INTRODUCCIÓN

ORIENTACIÓN A OBJ
- formados x datos (datos → operaciones) y comportamiento
- operaciones → datos → mod. estado obj
- interacción entre obj → envío de mensajes → obj (recibe, procesa, info)

APPS
- conjunto obj's → mensajes ↻ → resultados
- obj similares = agrupados clases
- 1 obj = 1 instancia de una clase

EJECUCIÓN
1. obj's se crean (según sea necesario)
2. Mensajes se mueven de un obj → obj (o del Usuario - obj)
3. Cuando obj → no necesario → eliminado → memoria libre

2. UML

Unified Modeling Language (unificado)

DEFINICIÓN → conj. herr. (documentar, modelar, construir) y sist. softw orientado a obj

ORIGEN → 3 métodos + usados en O.OBJ
- Grady Booch
- Ivar Jacobson
- Jim Rumbaugh

ESTÁNDAR DE LA INDUSTRIA

UTILIDADES
- facilita comunicación eq desarr. → lenguaje común
- documentar proceso desarr.
- posibilita especificar decisiones de análisis, diseño, implem. (precisa, compleja)

Conexión → leng prog (mediante) ingeniería (directa, inversa)

ELEMENTOS DIAGRAMA UML → **Estructurales** → nodos del grafo q definen el tipo diagrama
- **RELACIONALES** → arcos del grafo que establecen conex. entre estr.
- **NOTAS** → textos comentarios (ayudan comprender)
- **AGRUPACIONES** → uso → representar sist. grandes y facilitar su desarrollo x bloques.

TIPOS DIAGRAMA
- **ESTRUCTURALES**
  - clases
  - visión estática sist.
  - especif. diseño y obj → distribución física (y)
- **COMPORTAMIENTO**
  - conducta en tiempo de ejecución
  - pueden abordar → visión sist completo / instancias / objetos
  - casos de uso
  - casos de secuencia

HERRAMIENTAS
- LÁPIZ Y PAPEL
- CASE → entorno ventanas WYSIWYG (what you see → is what you get)
  - documentan
  - integran con entornos desarr.

DIAGRAMAS CASOS DE USO

DEFINICIÓN → diag. comportamiento → visualiza → interacción → roles
representa cómo los actores (usuario) operan en el sist → forma, tipo, secuencia } interacciones

PROPÓSITO → representar de manera general el funcion. de una App
→ visualizar funciones de un usuario

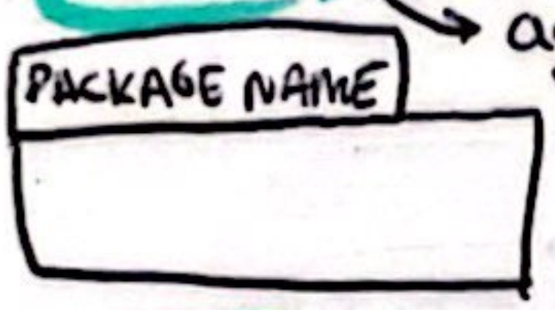
ELEMENTOS

ACTOR → rol/usuario
puede ser la extensión de otro

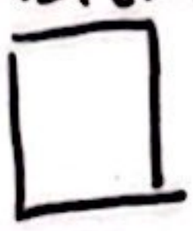
Ej: sist ventas → rol "Vendedor" → desempeñado → empleado, gerente

CASOS DE USO → operación/tarea específica → Ejecuta (Actor, Caso de uso)
nombre = VERBO

PAQUETE → elemento opcional → útil → diag. complejos
agrupar casos de uso



SISTEMA → definir alcance del caso de uso



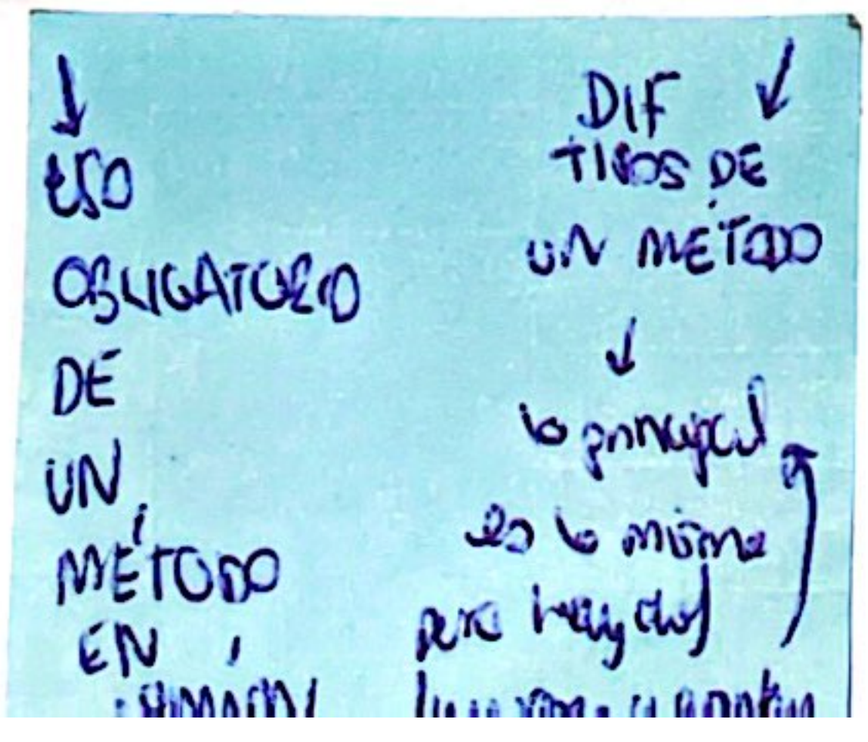
RELACIONES

ASOCIACIÓN → (Actor - Caso de uso) invocación

GENERALIZACIÓN

Uso (uses)
Conj de acciones similares en + de 1 caso de uso
"Función"
otro nombre → include

Herencia (extends)
1 caso de uso —(casó particular)→ de otro
Uso → casos de uso, actores



¿CÓMO CREAR?

1. IDENTIFICACIÓN DE LOS ACTORES → personas, sistemas, organizaciones

2. IDENTIFICACIÓN CASOS DE USO → de lo que los actores necesiten del sist.

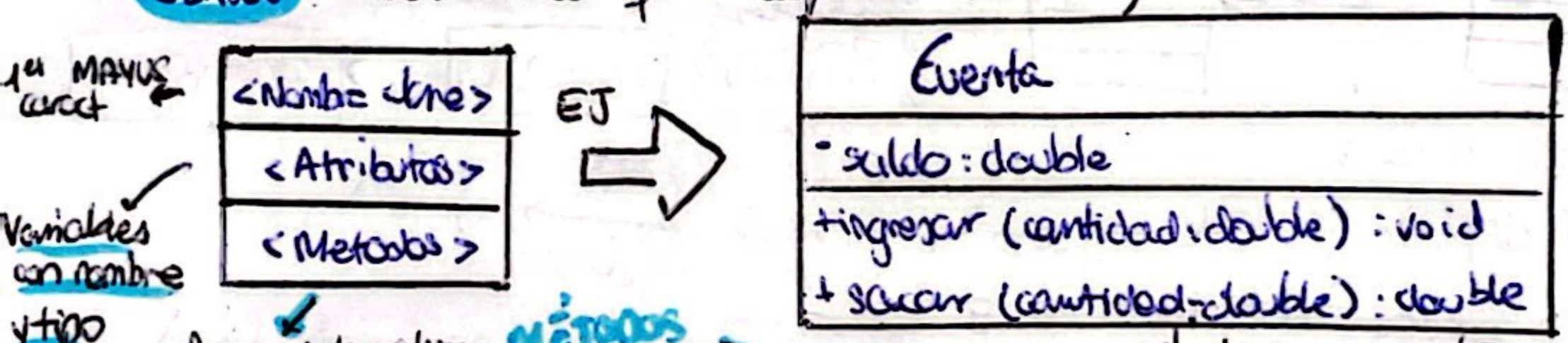
3. BUSCAR FUNCIONALIDAD COMÚN
   - Si varios casos de uso comparten funcionalidades similares
     - extraer fluidores → añadir → caso de uso
     - relación de inclusión (USES) → crearlos → mostrar q se llaman

4. FUNCIONES OPCIONALES/ADICIONALES
   - usar relación de extensión
   - el caso de uso base debe ser capaz de realizar una función x sí mismo, aunque no se llame caso de uso extensivo

DIAGRAMAS DE CLASE → sirve para visualizar las relaciones entre las clases que involucran el sist., atbos y comportamientos.

ELEMENTOS

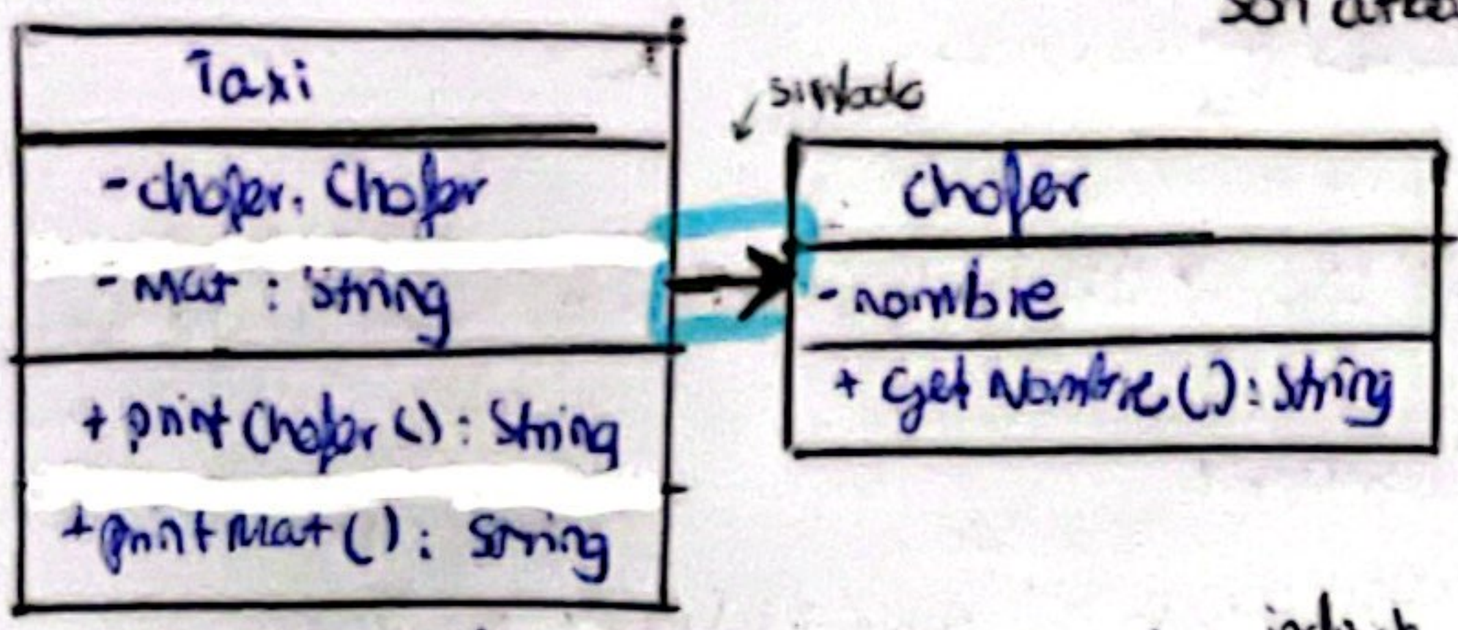
- **CLASES** → ud. básica que encapsula toda la info de un obj:



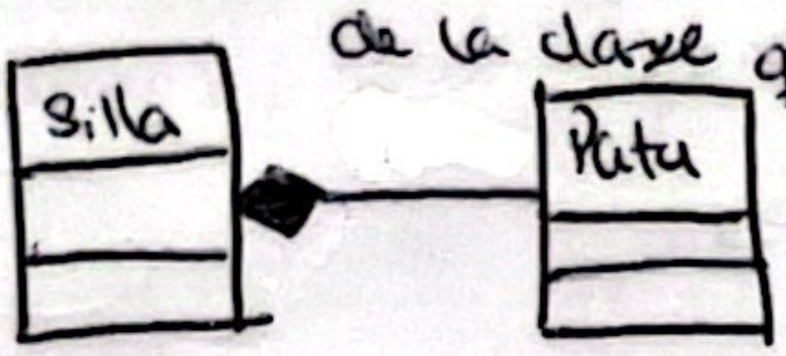
	clase	pqt	subclase	resto
+ public	S	S	S	S
- private	S	S	S	N
# protected	S	S	N	N
package	S	N	N	N

- **RELACIÓN ENTRE CLASES**
  - **CARDINALIDAD** → representa cuantos objs de 1 clase pueden estar involucrados en la relación con objs de otra.
    - 1:1
    - 1:n
    - 0:1
    - 0:n

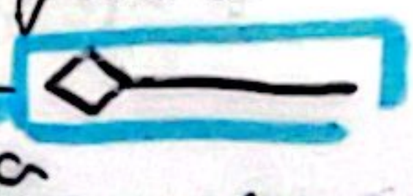
- **ASOCIACIÓN** → permite asociar objs que colaboran entre sí.
  1 obj / objs de 1 clase ⟶ de otra clase (son atbos)

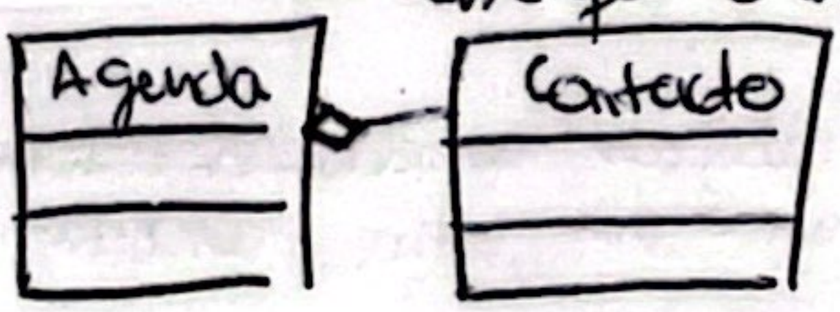


↳ **COMPOSICIÓN** → tiempo vida del obj incluido condicionado x el t. de vida del obj q lo incluye, ya que se crea (new) dentro de la clase que lo incluye. Rombo relleno



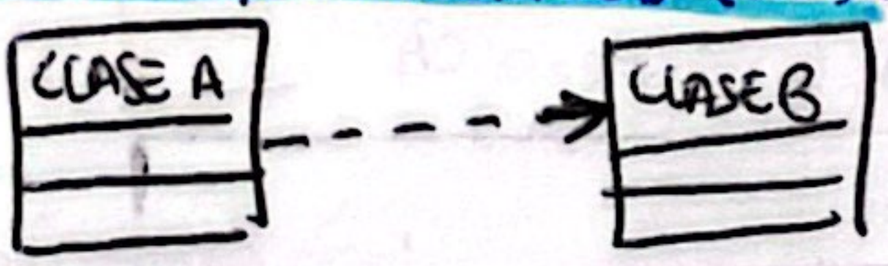
↳ **AGREGACIÓN** → t. de vida independiente del q lo incluye, no lo gestiona la clase que lo incluye. Rombo transparente



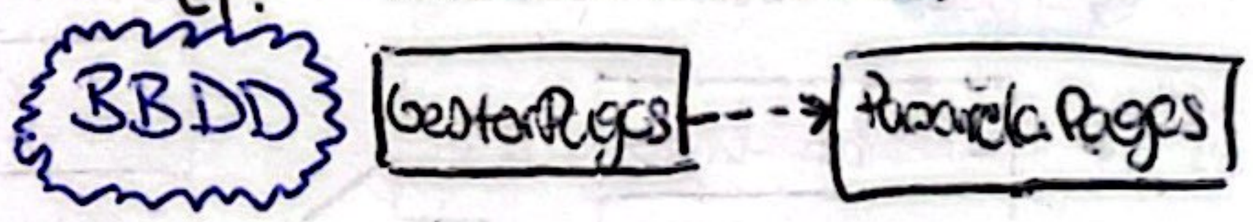


→ Cuando se destruye la agenda no se destruyen los contactos asociados (se crean "new" fuera de la clase Agenda)

- **DEPENDENCIA/INSTANCIACIÓN (USO)**: ⟶ Relación de uso entre 2 clases (una usa a la otra)



Ej:



  - la A usa a la B
  - La A depende de la B: 1 cambio clase B ⟶ (puede afectar) clase A
  - La A conoce existencia de B, pero la B no tiene pq conocer la A

- **HERENCIA (ESPECIALIZACIÓN/GENERALIZACIÓN)** ⟶

↳ subclase hereda → métodos, atributos de super-clase, además → características, atbos

protected → visibles → public