Notas de Diseño de Aplicaciones II

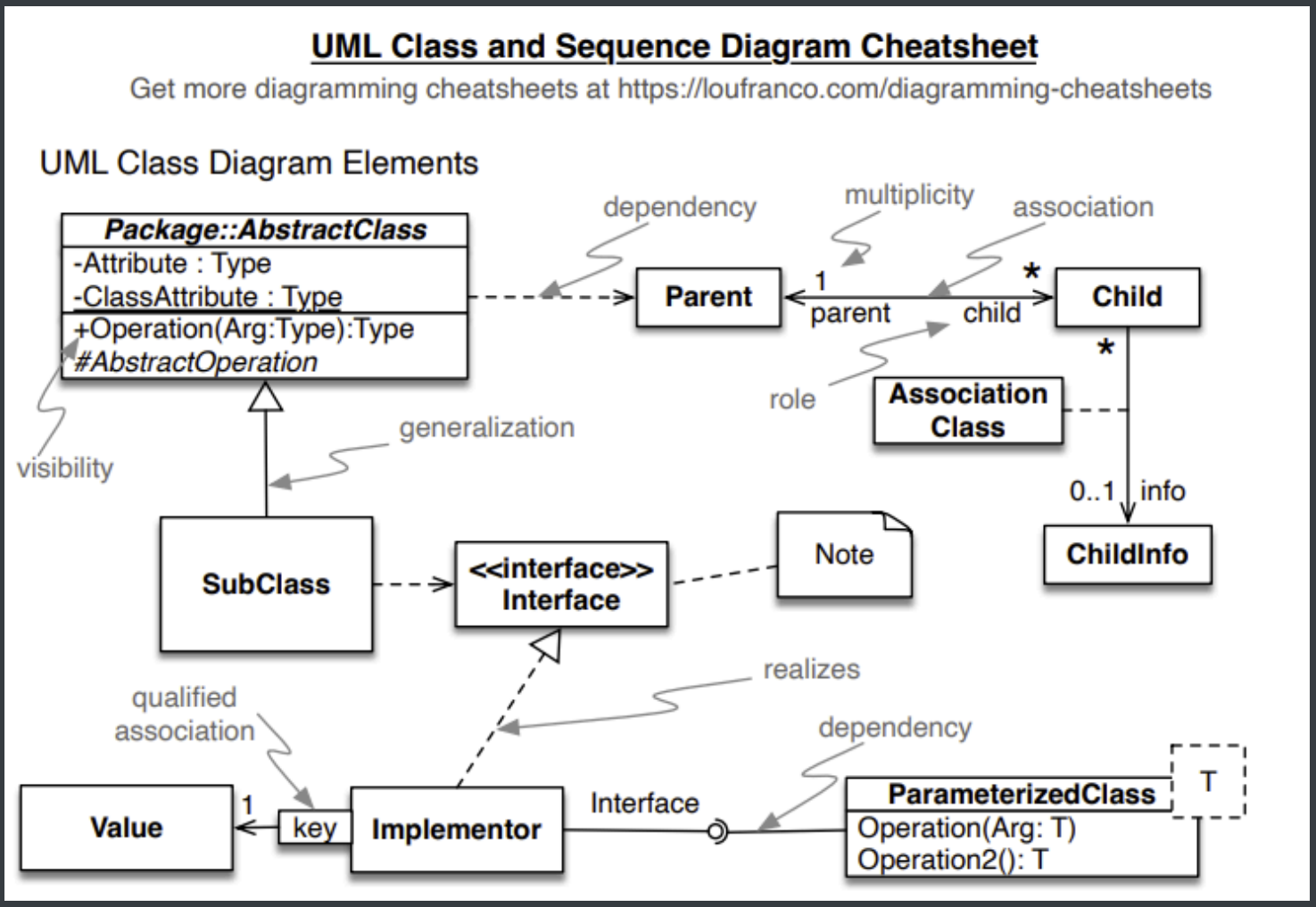
Profesor: Ignacio Valle
Correo: ignacio.valle@fi365.ort.edu.uy

Temario

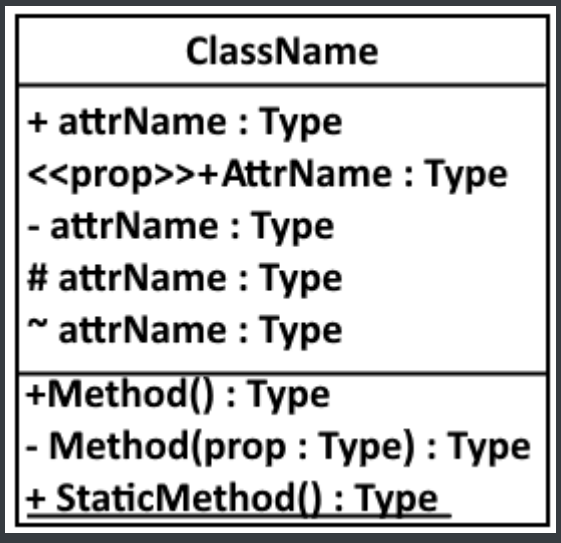
- Repaso D.A. I
- UML "Avanzado"
- Modelo "4 + 1" Arquitectura
- Calidad del Diseño
  - Aplicación de Patrones
  - Aplicación de Principios de Diseño (clases y paquetes)
  - Métricas de Diseño
- Introducción de Frameworks
- Arquitectura Web y Distribuidas (REST)

[Clase 1 [10/3/25] - Repaso D.A. I]

UML (Unified Modeling Language)



> Nota: Las clases **abstractas**, pueden indicarse mediante el uso de ***Fuente Italica***, o `<< abstract >>` prototype.



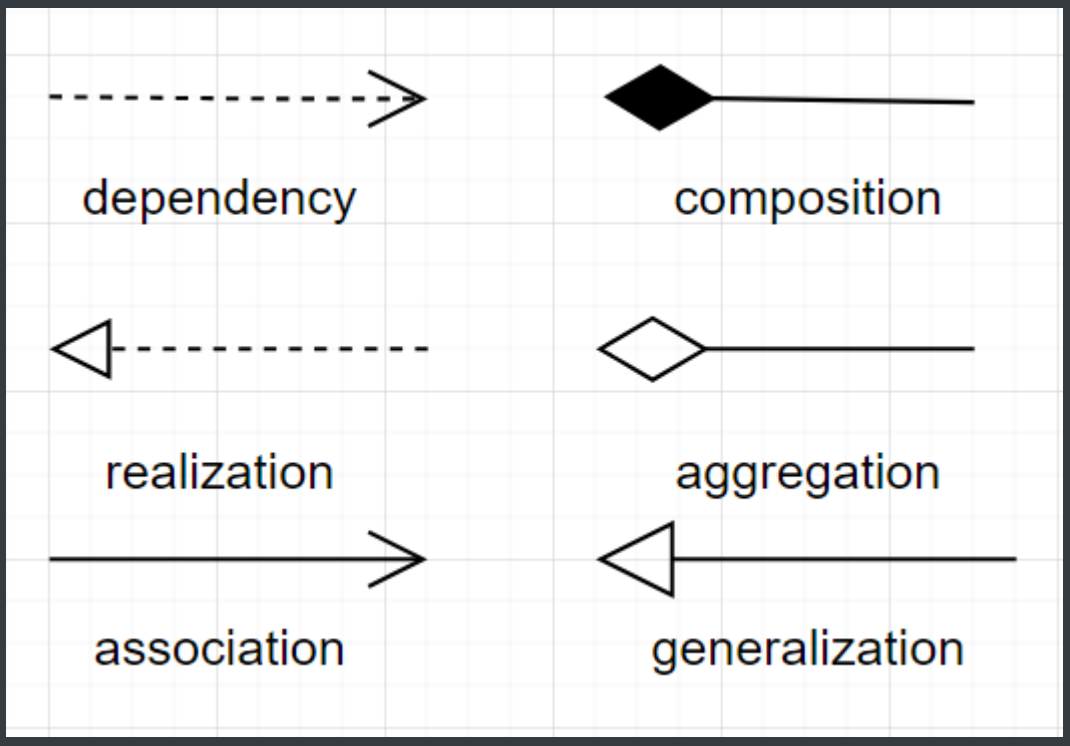
Visibilidad:

- `+` : **public**
- `-` : **private**
- `#` : **protected**
- `~` : **internal** (Solo para el Namespace)

> Nota: `<<prop>>` es una forma abreviada de escribir lo que se llama una "***Property***", es decir, un miembro de clase **privado**, con un **Getter** y **Setter públicos**.
>
> Por ejemplo:
>
> | **Student** |
> | --- |
> | - name : String |
> | + GetName()
+ SetName() |
>
> Sería lo mismo que:
>
> | **Student** |
> | --- |
> | <> Name |
> | |

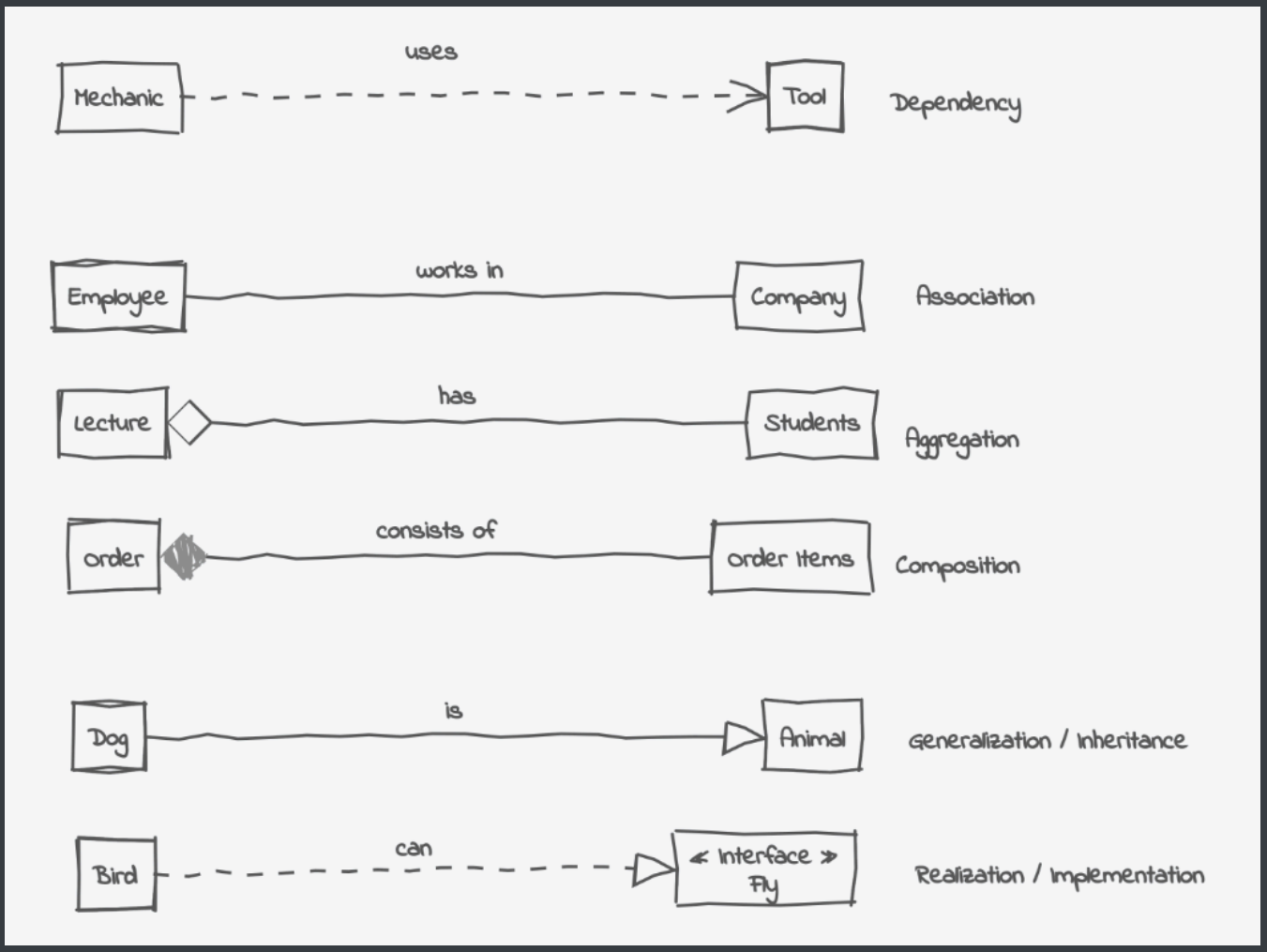
- `static` : Atributos o métodos subrayados indican que son propios de la clase, y no las instancias.

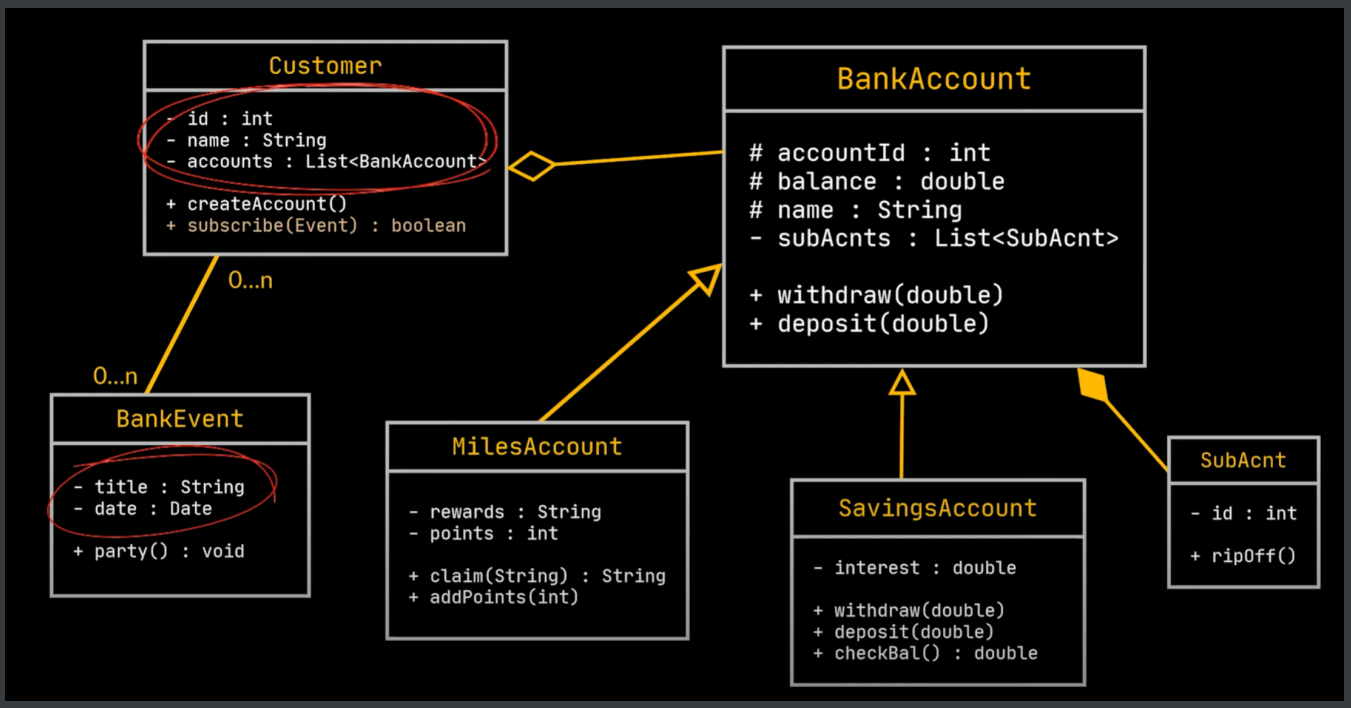
Relaciones entre Clases



When to Use Each Relationship

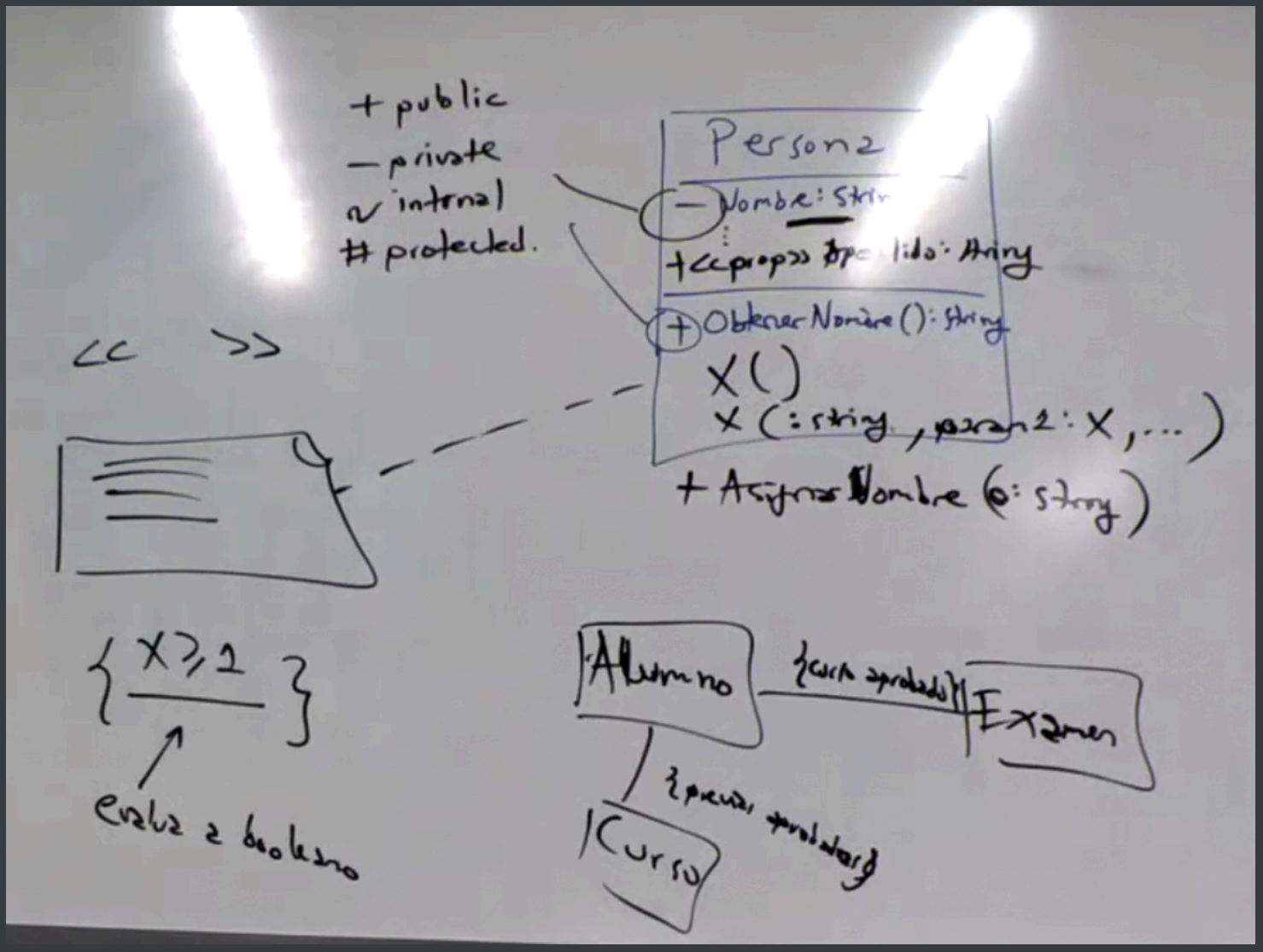
Relationship	When to Use
Dependency	When one class temporarily relies on another for functionality. (Ej.: Parámetro en un método)
Association	When two classes need to interact or communicate more permanently. (Ej.: Atributo de la clase)
Aggregation	When parts can exist independently of the whole. (Ej.: Atributo o lista de elemento que puede vivir independientemente)
Composition	When the parts' lifecycles are tied to the whole. (Ej.: Atributo o lista de elemento que no puede vivir independientemente)
Generalization	When defining an inheritance hierarchy. (Ej.: Padre de la clase)
Realization	When a class provides a concrete implementation for an interface. (Ej.: Implementación de la Interfaz - Esta relación no vá)


uses
Mechanic
Tool
Dependency
works in
Employee
Company
Association
has
Lecture
Students
Aggregation
consists of
order
order items
Composition
is
Dog
Animal
Generalization / Inheritance
can
Bird
« Interface »
Fly
Realization / Implementation


Customer
- id : int
- name : String
- accounts : List<BankAccount>
+ createAccount()
+ subscribe(Event) : boolean
BankAccount
accountId : int
balance : double
name : String
- subAcnts : List<SubAcnt>
+ withdraw(double)
+ deposit(double)
0...n
0...n
BankEvent
- title : String
- date : Date
+ party() : void
MilesAccount
- rewards : String
- points : int
+ claim(String) : String
+ addPoints(int)
SavingsAccount
- interest : double
+ withdraw(double)
+ deposit(double)
+ checkBal() : double
SubAcnt
- id : int
+ ripOff()

Estereotipos (extensiones de UML)

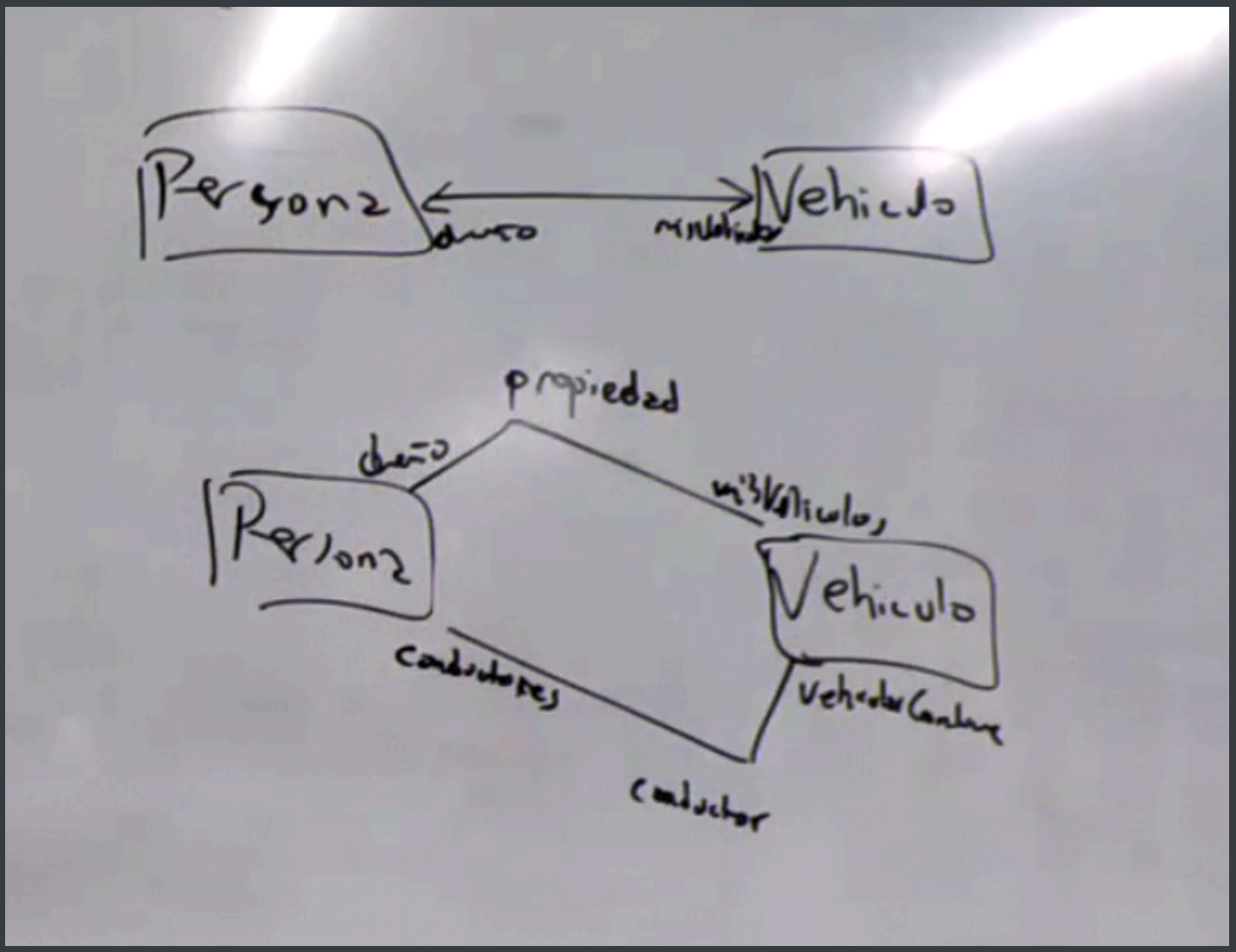
Existen herramientas que permiten implementar conceptos no cubiertos por el UML standard, y estos se conocen como estereotipos.



- `<< [concepto] >>` : Permite expresar un concepto particular (como los `props` en C#), originalmente utilizado con `<<interface>>` .
- `(Hoja con doblez)` : Simboliza una nota o comentario asociado a algún elemento.
- `{ [expresión booleana] }` : Representa una restricción (**"*constraint*"**) que debe cumplirse para que la relación suceda.

> En la imagen anterior, el diagrama inferior muestra dos ejemplos.
>
> 1. Para que el `Alumno` de `Examen` , debe tener el `{curso aprobado}`
> 2. Para que el `Alumno` se inscriba al `Curso` , debe tener las `{previas aprobadas}`

Rol, Cardinalidad, Navegabilidad, Nombre de Realción



Navegabilidad

El primer esquema de la imagen muestra una relación de **Asociación** con **Navegabilidad Bi-direccional** (indicado por la flecha en ambas direcciones).
Esto simboliza que la clase `Persona` posee una instancia de `Vehiculo`, y viceversa, `Vehiculo` posee una instancia de `Persona`.

Rol

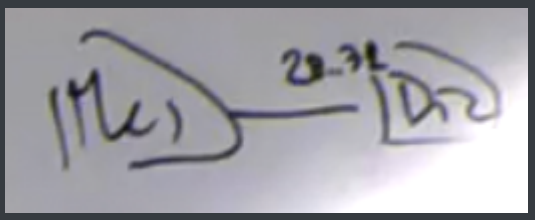
Debajo de cada flecha, podemos ver el **Rol** que cumple cada una. El `Vehiculo`, posee una instancia de `Persona` que representa al `-dueño`, mientras que `Persona` posee instancias de `Vehiculo` que representan `-misVehiculos`.

> Cabe mencionar que cuando indicamos un **Rol**, este representa un ***atributo*** y su ***visibilidad***, por lo tanto, no se repite dentro de la definición de la clase.

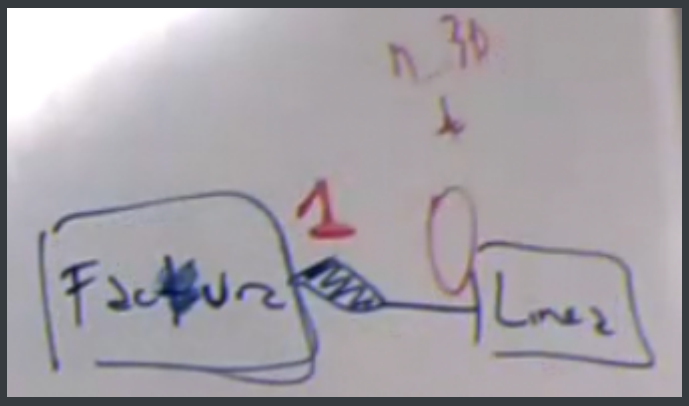
Nombre de Relación (Bi-direccionalidad diferenciada)

En el segundo esquema, podemos ver una relación cíclica, que si bien a simple vista parece una **Asociación Bi-direccional**, cada una posee **Roles** distintos. Podemos aclarar estas relaciones mediante un nombre ubicado en el medio de la misma (en este caso, `Propiedad` a la superior, y `Conductor` a la inferior). Estos nombres se denominan "***Nombres de Relación***".

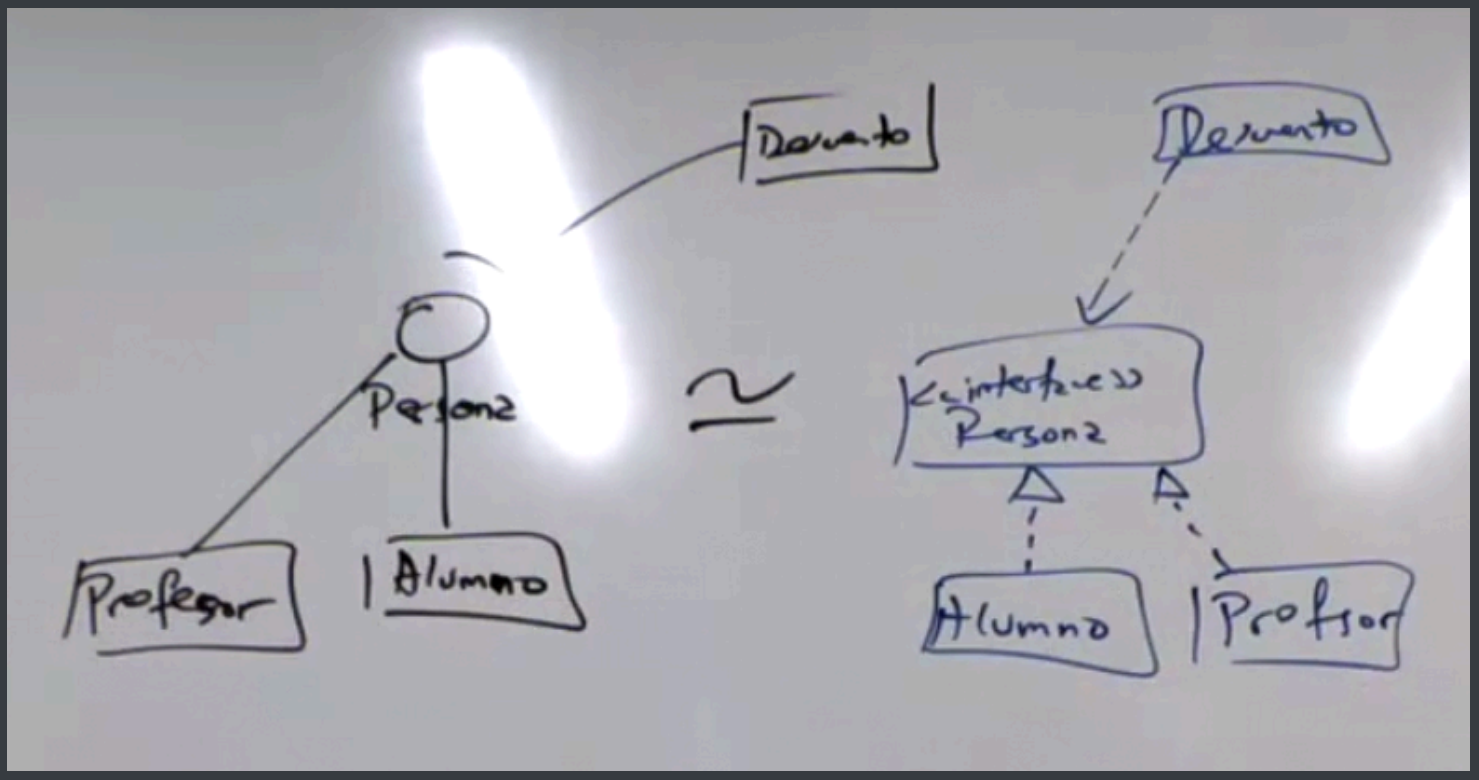
Cardinalidad



La cardinalidad, indica cuantas instancias de una clase pueden relacionarse con otras. En la imagen, se puede ver que cada `Mes`, puede tener entre `28...31` `Dias`. Pero estos valores pueden ser más estrictos (`1`, `0`, `5`, `n`, ...), así como rangos más flexibles (`*` = `0...n`, `1...n`, `2...6`, ...).

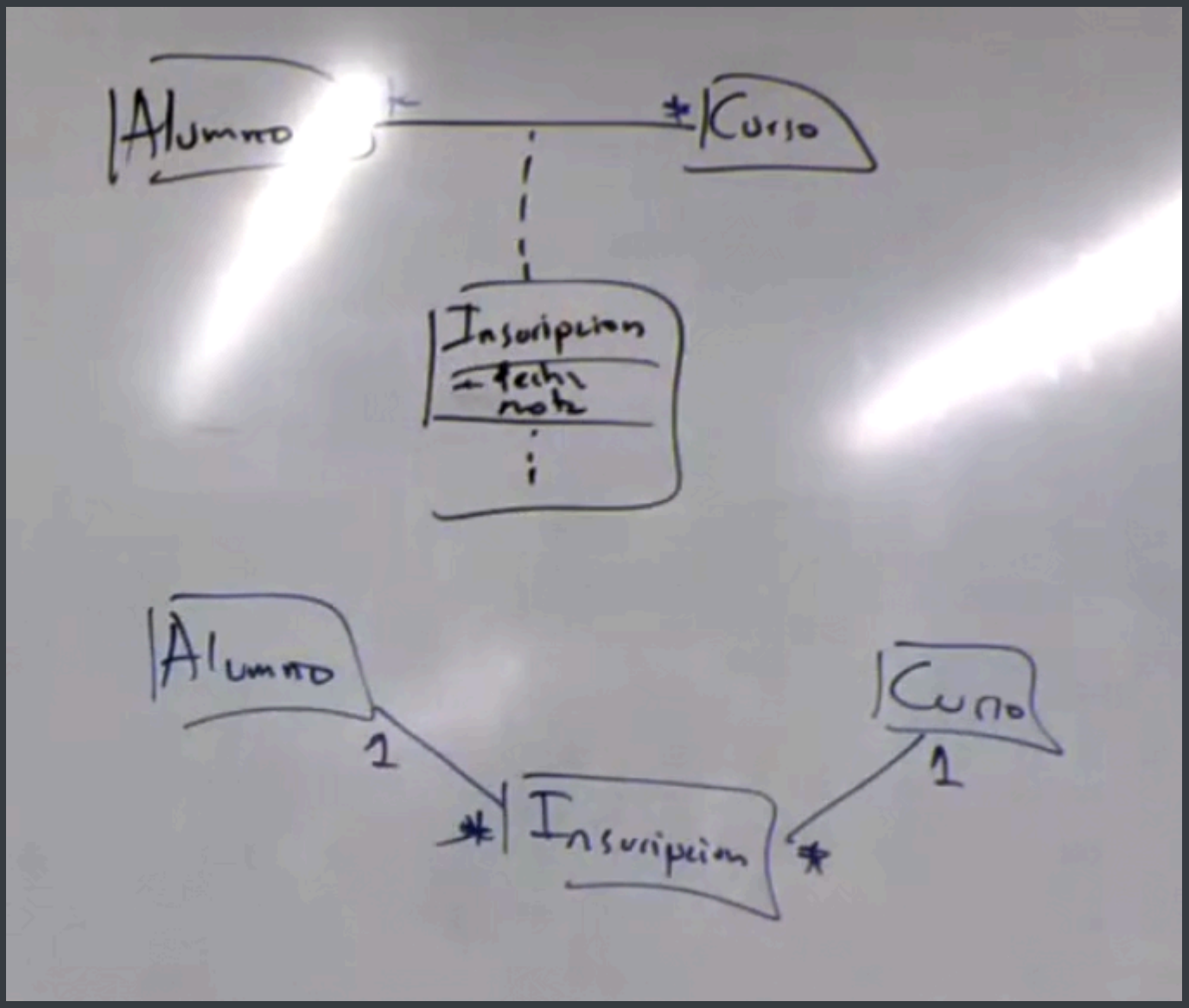
> La única restricción a la **cardinalidad**, es cuando hay "***relación de vida"***. Si una clase tiene "*relación de vida*" con otra, significa que depende de una única instancia, ya que de lo contrario, si la instancia de quien supuestamente depende cesara de existir, y esta no fuera eliminada, ya no sería una "*dependencia de vida*".
>
> 
>
> Un ejemplo de esto es `Factura` y `Linea`. Una `Factura` puede tener desde `0` a `n` líneas y seguir existiendo, mientras que cada línea solo puede pertenecer a una única `Factura` para considerarse "*dependiente*" de esta a nivel conceptual.

Representación de Interfases



> En la imagen, `Descuento`, posee una instancia de `Persona`, implementada por las clases `Alumno` y `Profesor`.
> Ambos esquemas representan lo mismo.

Relación con Atributos



Podemos hacer que ciertas relaciones se vean conceptualmente más adecuadas mediante el esquema de arriba, indicando que `n Alumnos`, están relacionados con `n Cursos`, pero para ello dependen de una `Inscripción`. Sin embargo, a nivel de código, se utilizaría su símil indicado en el esquema inferior.

Interfaz vs Clase Abstracta

Concepto de Clase Abstracta

Una **clase abstracta** representa una **abstracción parcial** de un concepto dentro de un sistema. Es un molde que define **qué es** un conjunto de objetos relacionados y proporciona cierta **estructura común**. Es útil cuando hay un vínculo jerárquico natural y queremos compartir código entre múltiples clases.

📌 **La clase abstracta es una base con una identidad clara.**
Ejemplo: Si definimos `Vehiculo` como abstracto, estamos diciendo que todos los vehículos comparten ciertas características y comportamientos (como moverse), pero cada uno los implementará de manera diferente (autos, motos, barcos).

🔷 **Piensa en la clase abstracta como una plantilla con una identidad específica y una estructura compartida.**

Concepto de Interfaz

Una **interfaz** representa una **capacidad** o **comportamiento**, pero no define qué es un objeto, sino qué puede hacer. Una interfaz no impone una estructura ni define atributos, solo establece un **contrato** que las clases deben cumplir.

📌 **La interfaz define un comportamiento sin imponer una identidad.**
Ejemplo: `Volador` es una interfaz porque diferentes cosas pueden volar (pájaros, aviones, superhéroes), pero no todas tienen una relación jerárquica entre sí.

🔷 **Piensa en la interfaz como un conjunto de reglas que cualquier clase puede seguir sin importar su origen.**

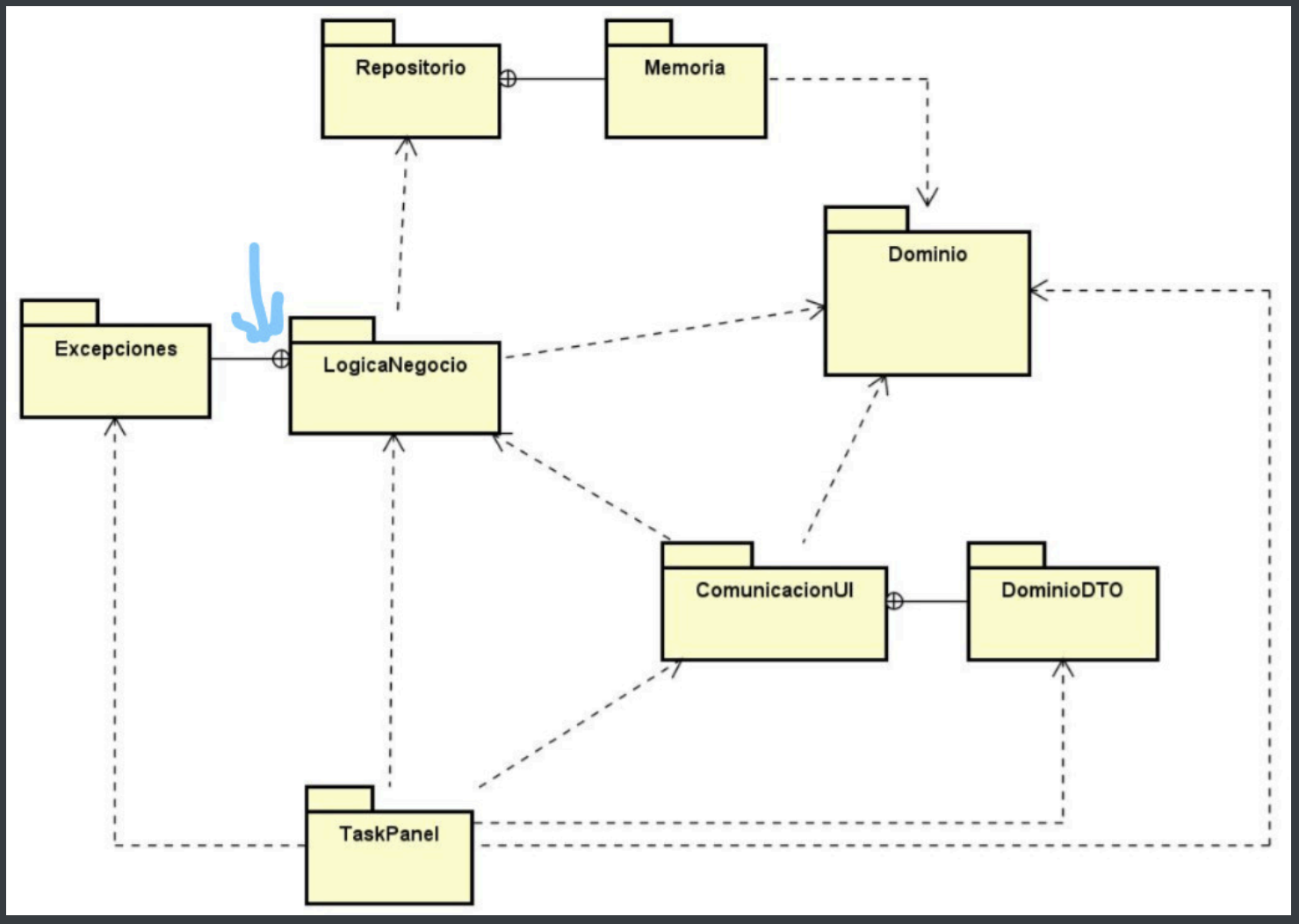
Diferencia fundamental

- **Clase abstracta**: Define una base conceptual de un conjunto de objetos similares con una identidad clara.
- **Interfaz**: Define capacidades independientes que pueden ser adoptadas por cualquier clase, sin necesidad de compartir una misma estructura.

Cuando decides entre una clase abstracta y una interfaz, pregúntate:

- ¿Estoy definiendo **qué es** algo? → Usa una **clase abstracta**.
- ¿Estoy definiendo **qué puede hacer** algo? → Usa una **interfaz**.

Diagrama de Paquetes



Es simplemente marcar que paquete depende de cual, y con la relación señalada en la imagen de arriba, significa que un paquete ("*Excepciones*") está contenido en otro ("*LógicaNegocio*").