NOTACIÓN UML: CLASES DE DISEÑO

OBJETIVOS

- Comprende las diferencias entre clases y objetos.
- Aplica la notación UML para elaborar diagramas de clases de diseño.
- Determina cuándo aplicar cada uno de los principios de diseño.

1

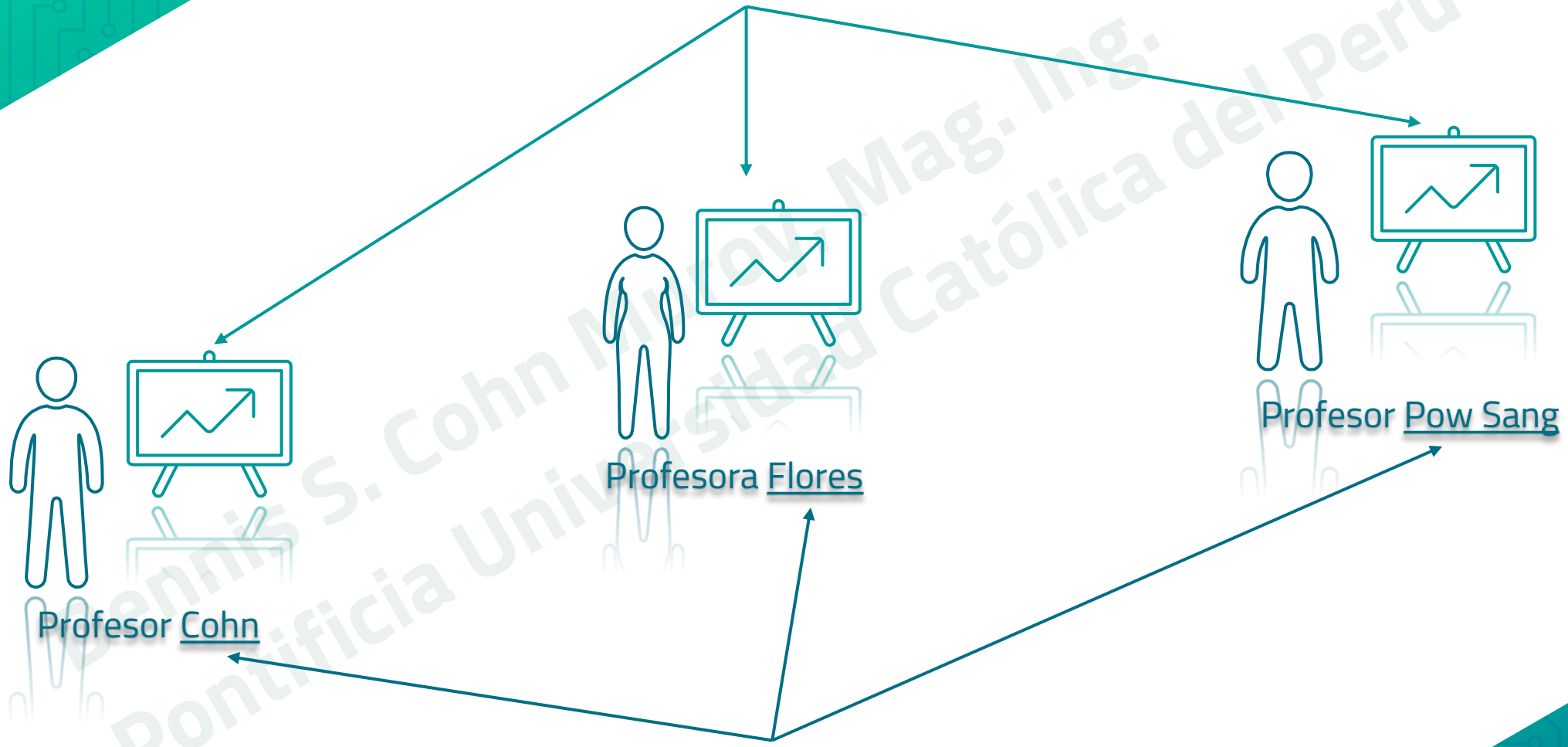
DEFINICIONES

Sobre Clases y Objetos

¿QUÉ ES UN OBJETO?

- Entidad que tiene una naturaleza de tipo:
  - Físico (ejemplo: persona, automóvil, casa ...)
  - Conceptual (ejemplo: proceso químico, trámite administrativo)

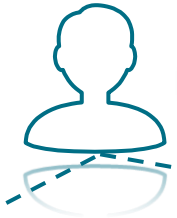
Elemento común: enseñan el mismo curso



Independientes: propio identificador / identidad

¿QUÉ ES UNA CLASE?

- Descripción o taxonomía que agrupa un conjunto de objetos que tienen propiedades comunes.
- Estas propiedades comunes se denominan atributos.



Docente

Nombres
Apellidos
Fecha de Nacimiento

Atributos

Profesor Cohn

Profesora Flores

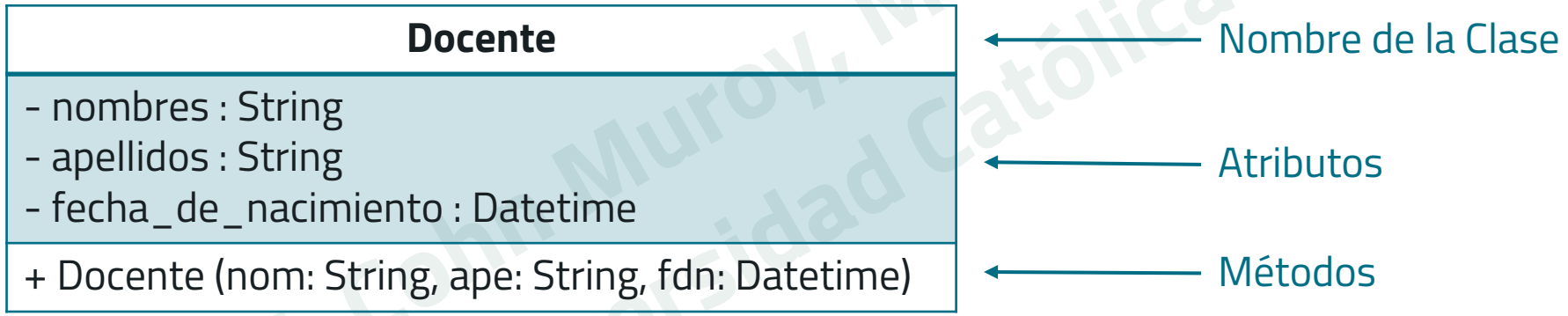
Profesor Pow Sang

2

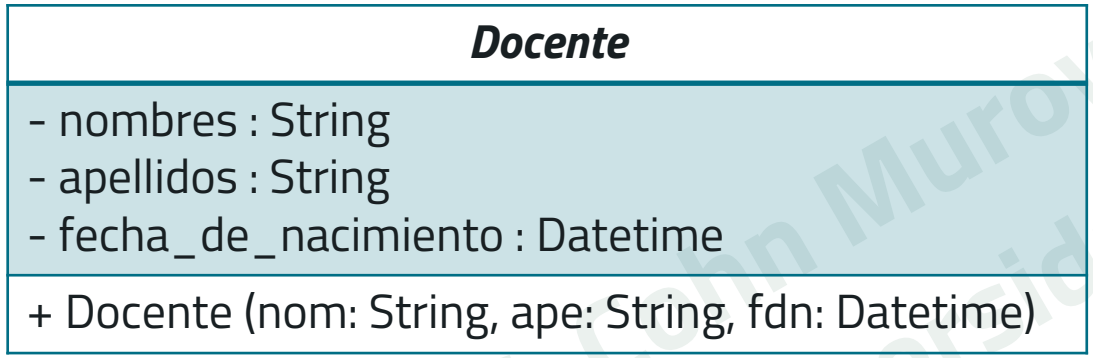
CLASES DE DISEÑO

Notación en UML

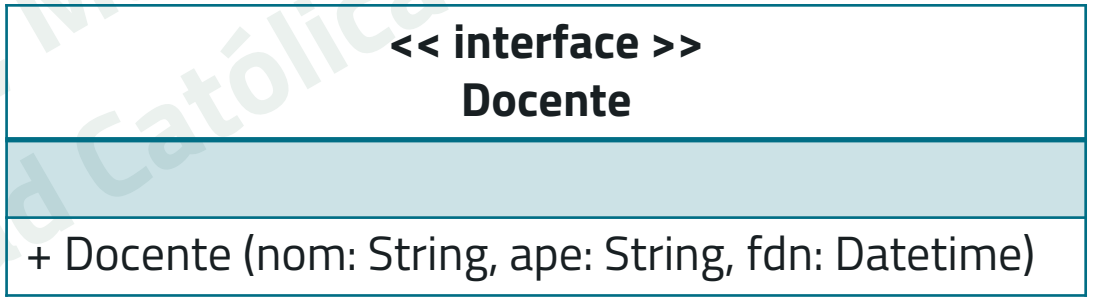
¿CÓMO REPRESENTAMOS LAS CLASES EN UML?



¿CÓMO REPRESENTAMOS LAS ABSTRACCIONES?



Clase Abstracta



Interfaz

NOTACIÓN DE ATRIBUTOS

Visibilidad	Nombre del Atributo	:	Tipo de Dato	= Valor por defecto (opcional)
público (+) privado (-) protegido (#) paquete (~)	Nombre del Atributo (estático)		string int double etc.	

Ejemplos

identificador : int = 0 → atributo protegido de tipo int con nombre “identificador” y valor inicial “0”.

- nombres : string → atributo privado de tipo string con nombre “nombres”.

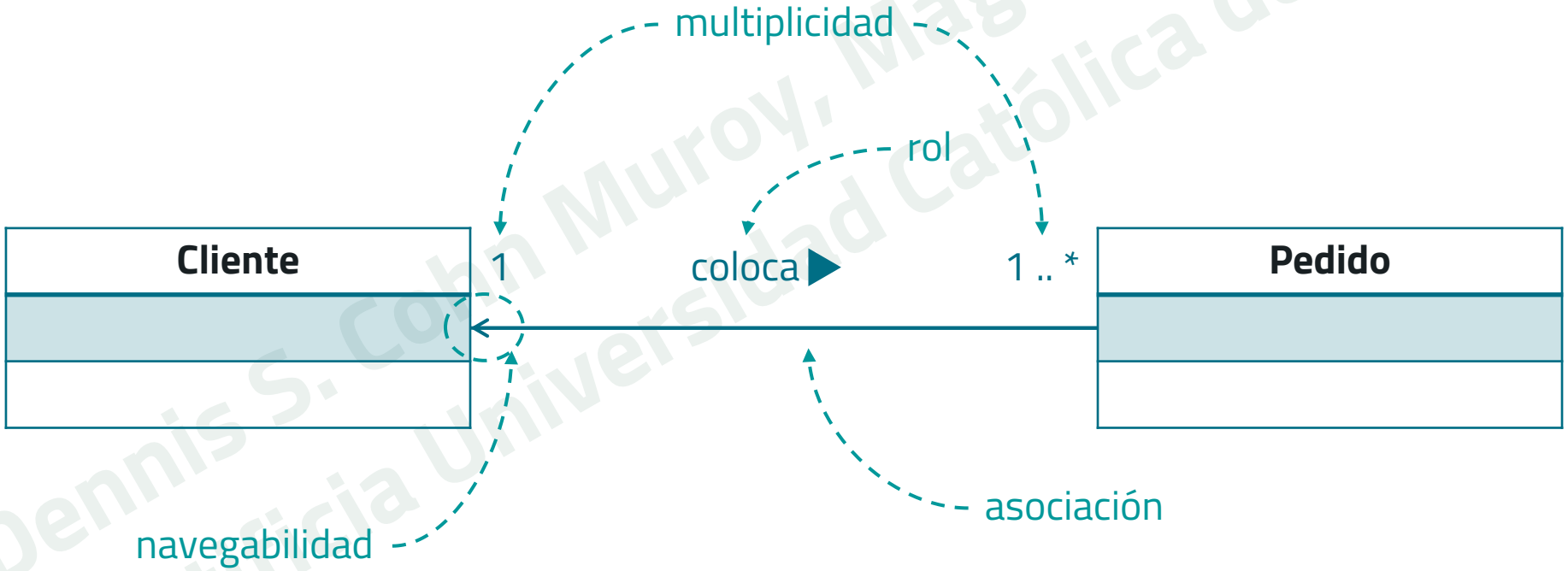
NOTACIÓN DE MÉTODOS

Visibilidad	Nombre del Método	(Parámetros)	:	Tipo de dato de salida
público (+) privado (-) protegido (#) paquete (~)	Nombre del Método (estático)	Nombre : tipo de dato		string, int, double, etc. (Omitir en el constructor) (void es opcional)

Ejemplos

+ iniciarSesión () → método público para iniciar sesión en el sistema.
+ obtenerInstancia () : ConectorBD → método estático público que retorna una instancia del ConectorBD.
- validarDatos (nombres : string, apellidos: string) : boolean → método privado que valida los nombres y apellidos; retorna un valor lógico.

RELACIONES ENTRE CLASES

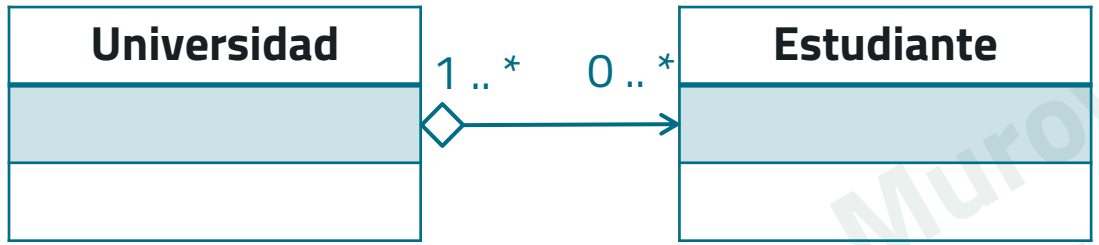


(*) Contexto de Ventas

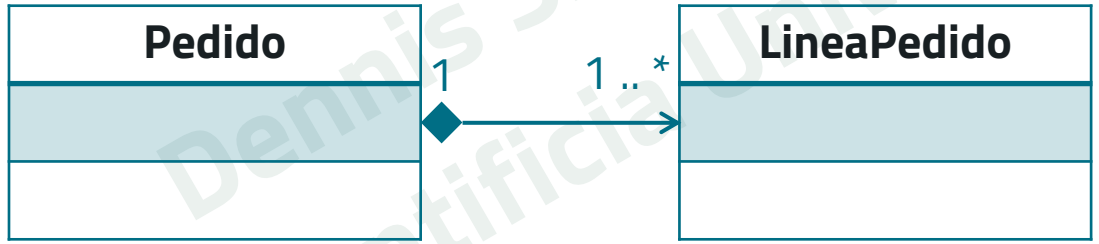
LA NAVEGABILIDAD PUEDE SER HACIA CUALQUIER DIRECCIÓN

De izquierda a derecha, de derecha a izquierda o en ambos sentidos.

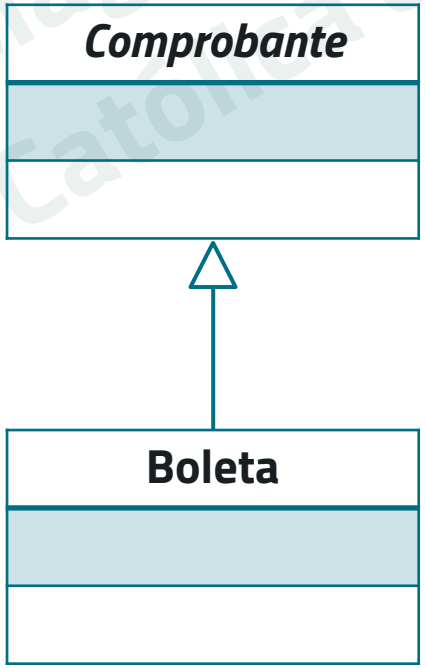
TIPOS DE RELACIONES



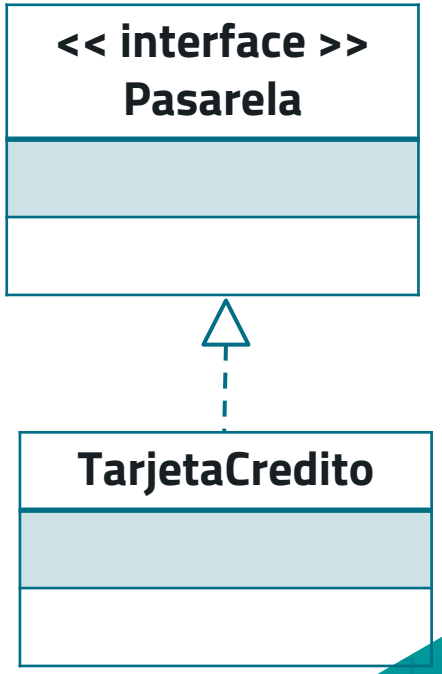
Agregación



Composición



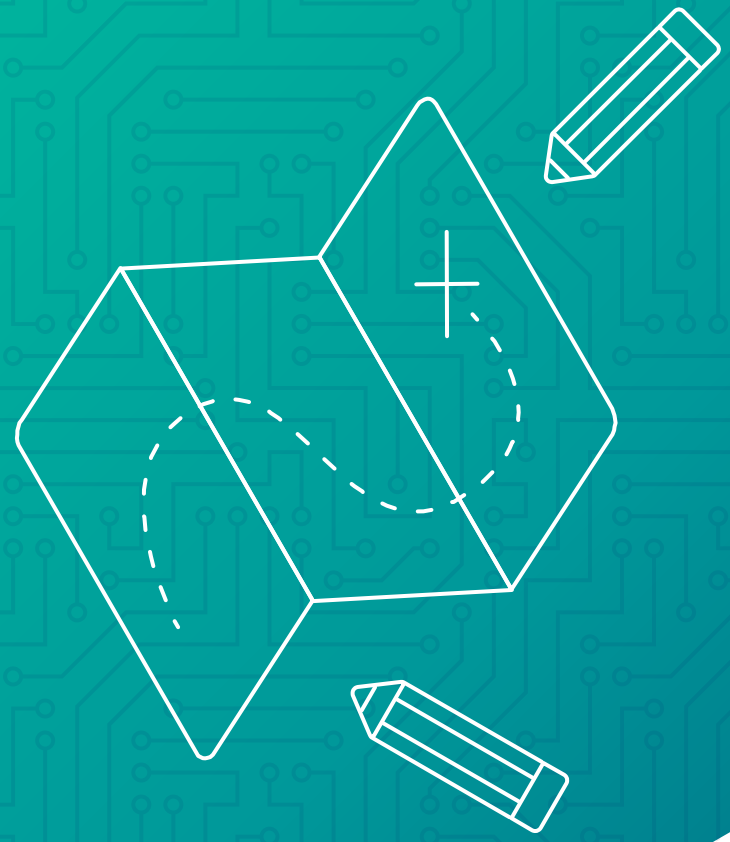
Herencia



Implementación

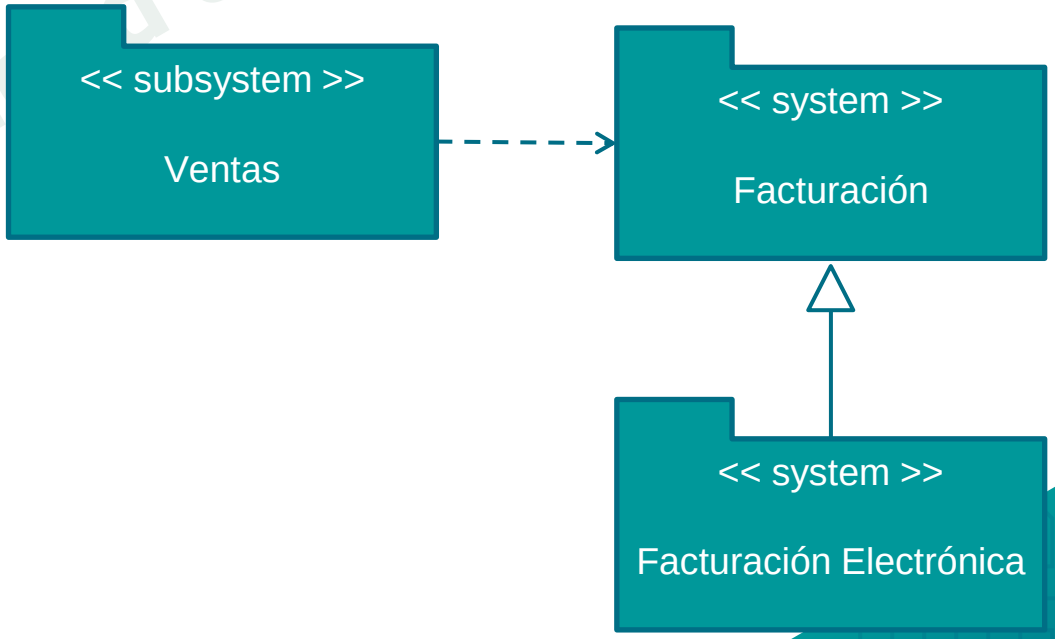
OTROS ELEMENTOS A INCLUIR EN EL DIAGRAMA

No olvidar incluir otras clases como: ventanas, controladores (o gestores), persistencia de objetos, otras clases propias de una estrategia de diseño.



¿CÓMO AGRUPAMOS CLASES?

- Las Clases se agrupan en Paquetes.
- Los Paquetes agrupan clases lógicamente relacionadas:
  - A nivel técnico.
  - Por dominio o contexto.



¿POR QUÉ ES IMPORTANTE ESTE DIAGRAMA?

- Describe la estructura del sistema a implementar a través de:
  - Clases,
  - Atributos,
  - Métodos, y
  - Relaciones entre clases.
- La implementación de las clases debe darse tal cual se ha elaborado en este diagrama.

3

PRINCIPIOS DE DISEÑO

Consideraciones durante el diseño ...

GUÍAS

(NO DIRECTIVAS)

PARA UN BUEN DISEÑO

El software a implementar debe ser entendible, flexible (evolutivo) y mantenible.

ABSTRACCIÓN & SIMPLICIDAD

- ABSTRACCIÓN
  - Los bloques de construcción deben depender de abstracciones
    - Interfaces.
    - Clases abstractas
- SIMPLICIDAD
  - KISS (Keep It Small & Simple) – Diseña solo para los requerimientos identificados (presentes y futuros).
  - DRY (Don't Repeat Yourself) – No duplicar código.

INTEGRIDAD CONCEPTUAL & ESPERAR ERRORES

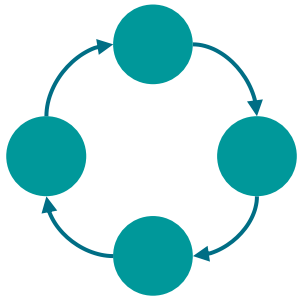
- INTEGRIDAD CONCEPTUAL
  - Tareas similares deben manejarse de una misma forma.
    - UNIX: Todo es un fichero.
    - LISP: Todo es una lista.
- ESPERAR ERRORES
  - Resiliencia (estable, confiable, disponible).
  - Robustez (tolerancia a fallos).

MODULAR

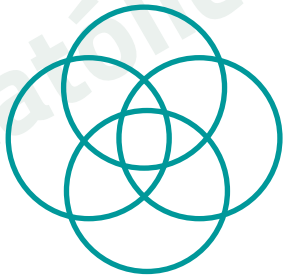
- MODULAR
  - Principio de Bloques de construcción.
  - División del sistema en bloques de construcción.
    - Encapsulan responsabilidades.
    - Exponen únicamente interfaces bien definidas.
    - Pueden ser mantenidos de forma independiente.
    - Pueden reemplazarse por módulos con las mismas interfaces.

MODULAR – ACOPLAMIENTO & COHESIÓN

Acoplamiento: Dependencias entre componentes.

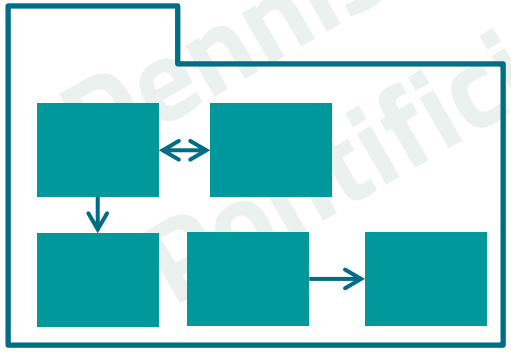


Bajo

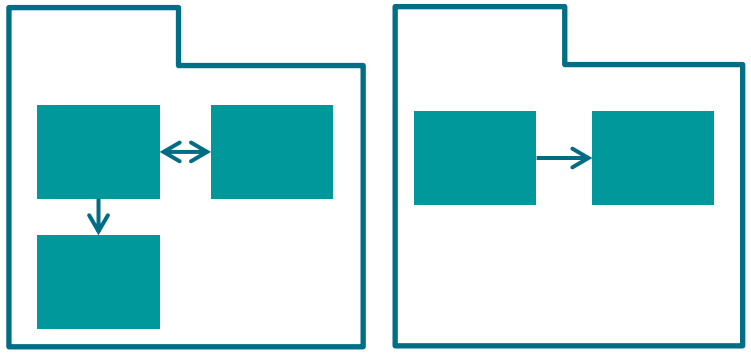


Alto

Cohesión: Relación de los elementos dentro de un componente.



Baja

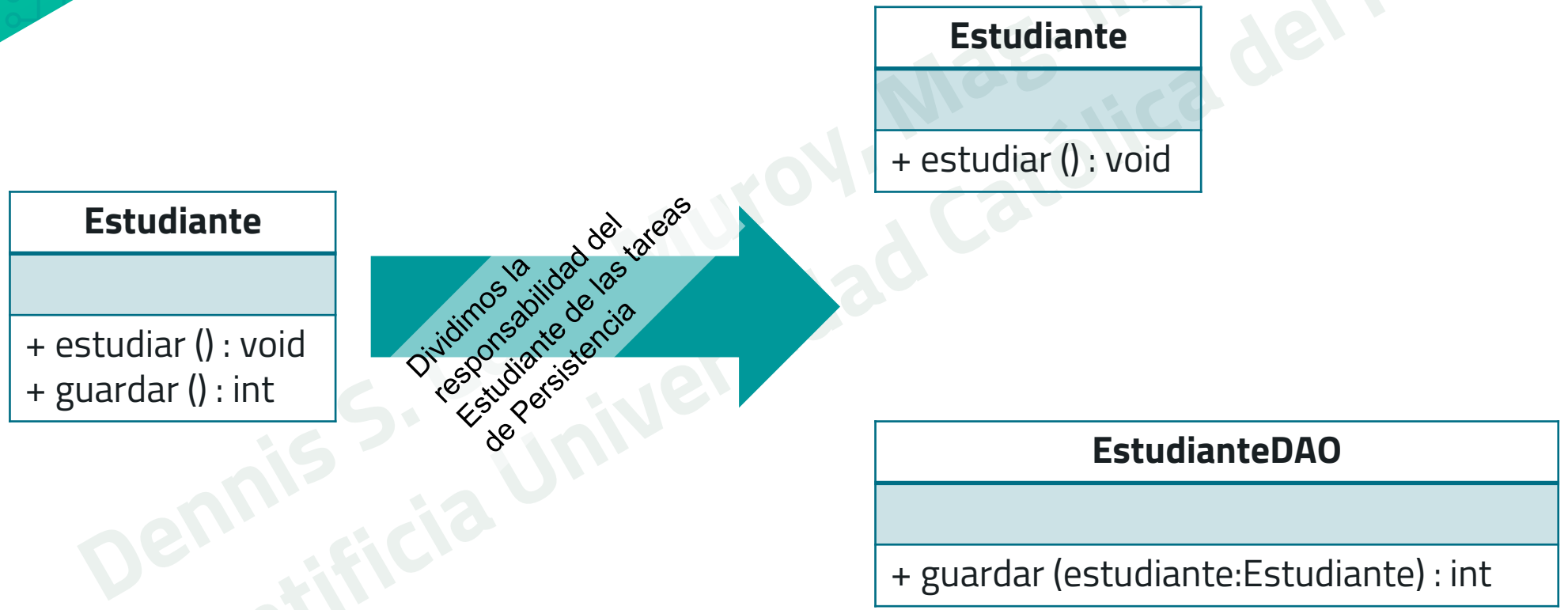


Alta

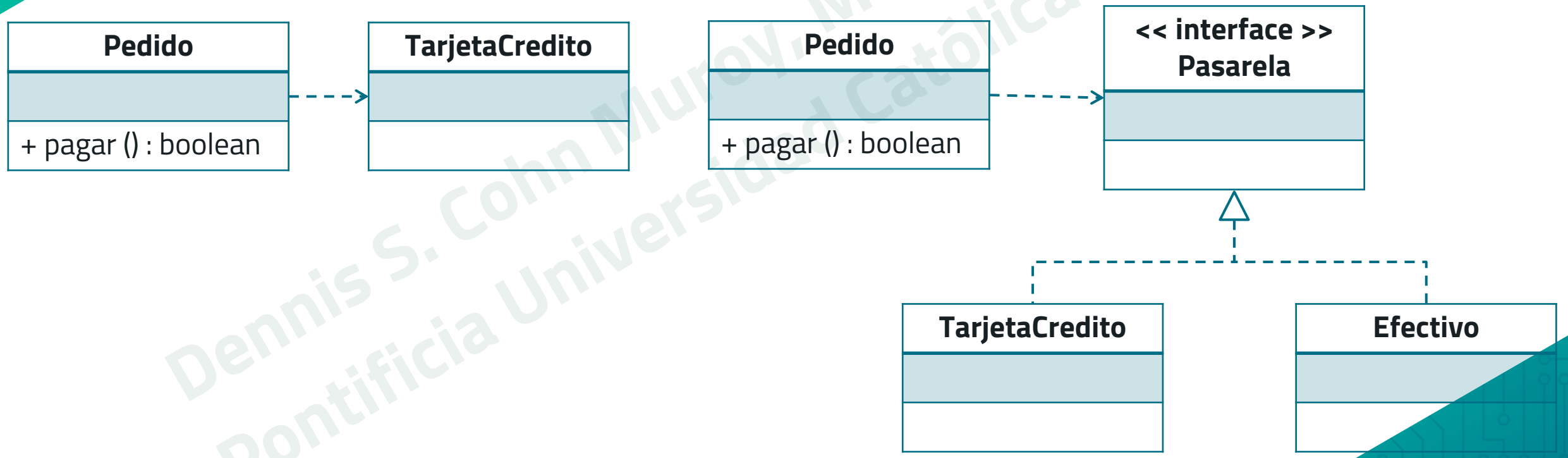
MODULAR - PRINCIPIOS S.O.L.I.D.

- S.O.L.I.D.
  - Single Responsibility: Un componente es responsable de una tarea.
  - Open/Closed: Un componente debe ser extensible; mas no modificable.
  - Liskov substitution: Subclases deben poder reemplazar Superclases.
  - Interface segregation: Contar con interfaces específicas por cada cliente.
  - Dependency Inversion: Dependencias deben darse hacia abstracciones.

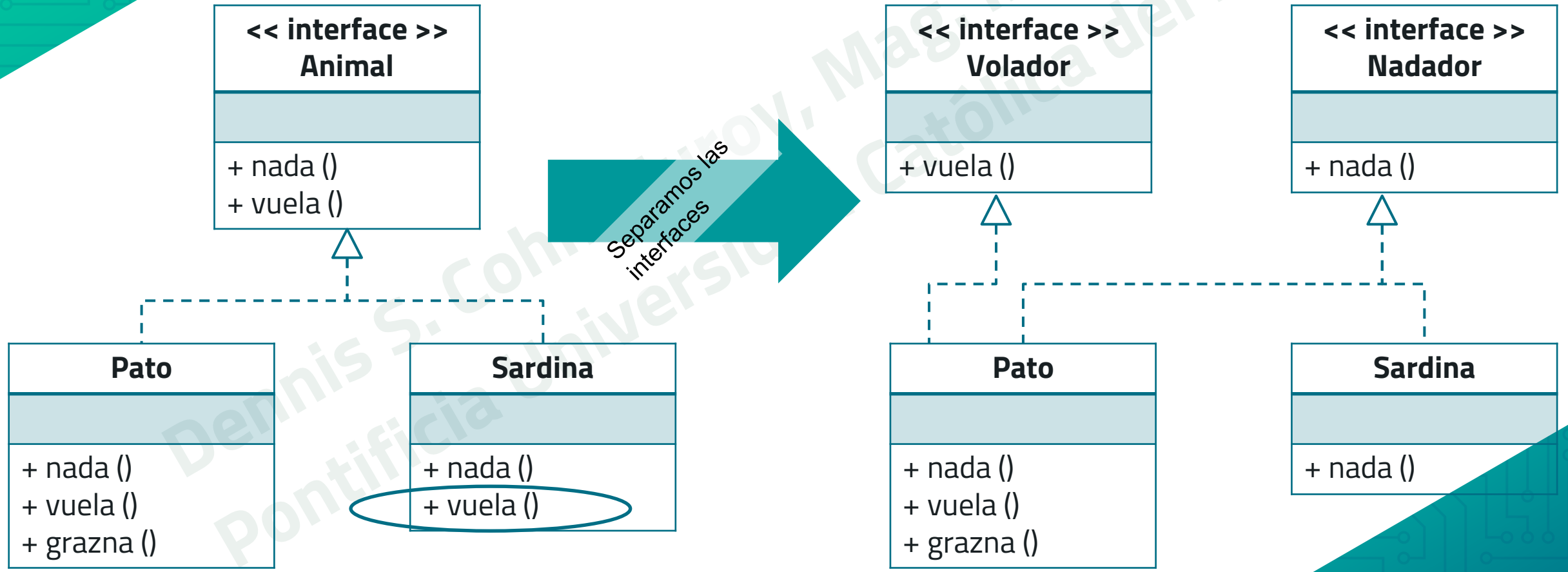
S.O.L.I.D: SINGLE RESPONSIBILITY



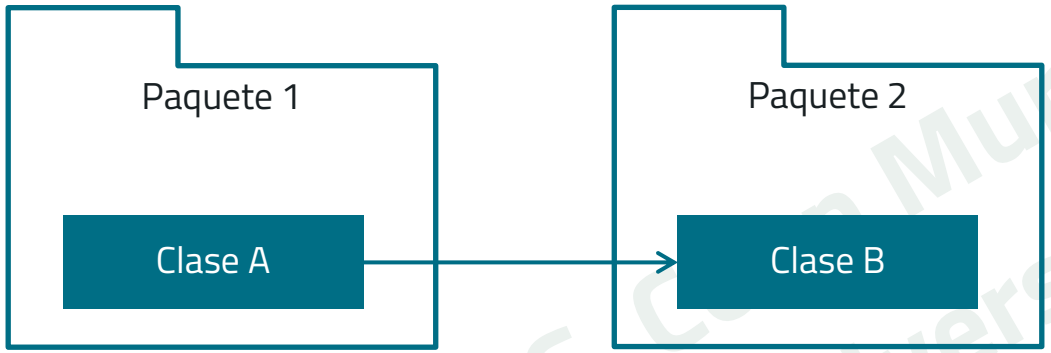
S.O.L.I.D: OPEN / CLOSED



S.O.L.I.D: INTERFACE SEGREGATION

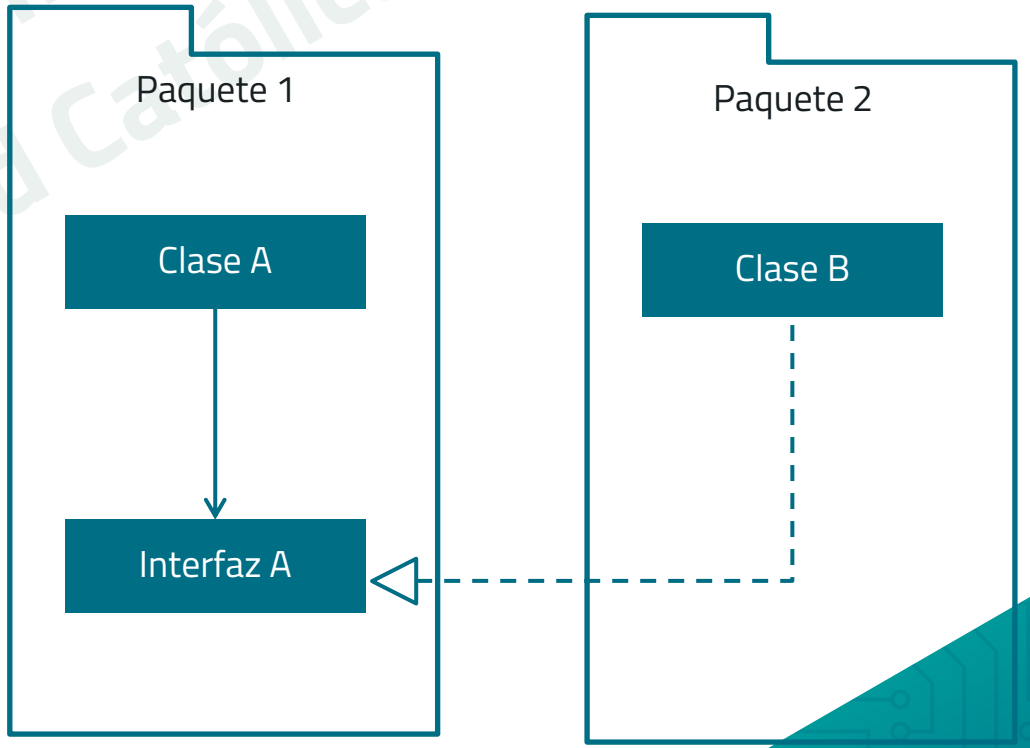


S.O.L.I.D: DEPENDENCY INVERSION



Acoplamiento fuerte; baja mantenibilidad.

Acoplamiento débil; mayor mantenibilidad.



4

CASO

Resolver el siguiente caso propuesto ...

SISTEMA DE RESERVA DE LIBROS

Diseña un sistema de reservas para una biblioteca universitaria. El sistema debe permitir a los estudiantes y profesores reservar libros físicos y recursos digitales.

- Los libros físicos que están disponibles pueden ser reservados. Estos tienen un título, un autor y un género.
- La biblioteca cuenta con recursos digitales como e-books, revistas electrónicas o bases de datos. Estos tienen un nombre, un enlace y un tipo de licencia.
- Los usuarios de la biblioteca son tanto estudiantes como profesores. Cada uno de ellos tiene un nombre y un número de identificación.
- Un usuario puede realizar una reserva para un libro o recurso digital específico. La reserva incluye información de fecha y estado (pendiente, confirmada, cancelada).

SISTEMA DE GESTIÓN DE PROYECTOS

Diseña un sistema de gestión de proyectos que permita a los equipos colaborar en la planificación, seguimiento y ejecución de proyectos. El sistema debe considerar lo siguiente:

- Se cuentan con varios proyectos, cada uno tiene un nombre, descripción y fecha de inicio.
- El proyecto está conformado por tareas. Cada una tiene un nombre, estado (pendiente, en progreso, completada) y está asignada a un miembro del equipo.
- Cada proyecto está asignado a un equipo que es un grupo de personas que trabajan de forma conjunta. Cada equipo tiene un nombre, un líder y sus miembros.
- Durante el proyecto se tienen reuniones en donde se discute el avance del proyecto. Las reuniones se agendan para una fecha y hora; e incluyen un listado de temas a tratar.

5

REFERENCIAS

BIBLIOGRAFÍA

- Sommerville, I. (2011). Software engineering (ed.). America: Pearson Education Inc.
- Starke, G., & Lorz, A. (2021). Software Architecture Foundation: CPSA Foundation® Exam Preparation (2nd ed.). Van Haren.

Créditos:

- Plantilla de la presentación por SlidesCarnival
- Fotografías por Unsplash
- Diseño del fondo Hero Patterns