PARADIGMAS DE PROGRAMACIÓN

Licenciatura en Informática
Programador Universitario

UNIDAD VII
PARADIGMAS DE PROGRAMACIÓN
Principios de Diseño Orientado a Objetos

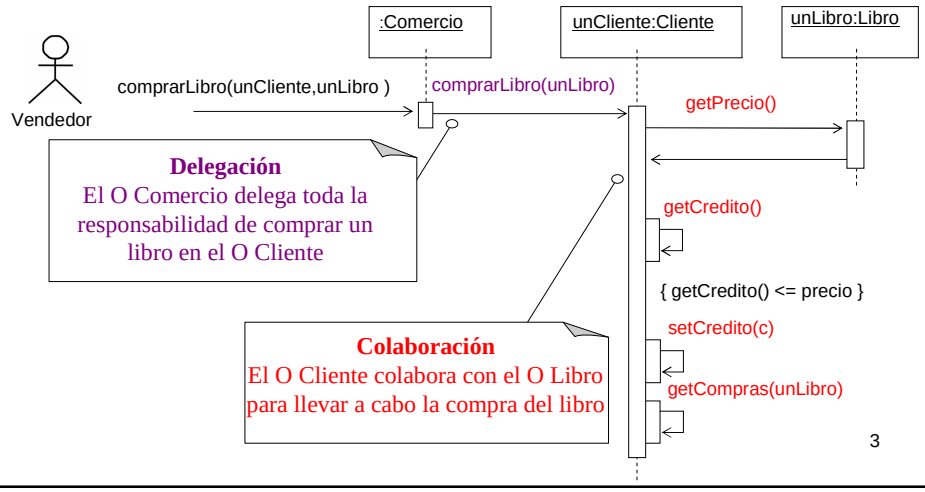
Diseño OO

Pautas de Diseño

- **ALTA Cohesión y BAJO Acoplamiento**
  - **Cohesión** → *Grado en el que las responsabilidades de una componente forman una unidad significativa*
  - **Acoplamiento** → Grado de dependencia de la implementación de un O
- **Herencia vs Asociación** — A diferencia de la asociación, la herencia trae implícitamente asumido que una subclase es en realidad un subtipo
- **Delegación vs Colaboración** — La diferencia se basa en **CUAL** es el O que realiza una determinada tarea
  - **Delegación** → Transfiere el control de la ejecución a otro O
  - **Colaboración** → El O delega parte de la responsabilidad en uno o más O, pero nunca pierde el control de la ejecución

Delegación vs Colaboración

Diagrama de Secuencia
de la compra de un Libro



3

Herencia vs Asociación

HERENCIA para definir una taxonomía fundamental
ASOCIACIÓN para combinar la funcionalidad existente

HERENCIA	ASOCIACIÓN
Reutilización a nivel de clase	Reutilización a nivel de instancia u objeto
Reutilización de caja blanca	**Reutilización de caja negra**

4

Herencia vs Asociación

HERENCIA	ASOCIACIÓN
Viola el principio de encapsulamiento	**No rompe el encapsulamiento**
•La herencia impone parte de la representación física a las subclases. Para saber que operaciones se pueden aplicar al nuevo tipo de datos se debe examinar las clases a lo largo de la jerarquía de herencia	•Es más simple, indica claramente que operaciones se pueden aplicar en una estructura de datos particular

5

Herencia vs Asociación

HERENCIA	ASOCIACIÓN
•Cambios en la superclase pueden afectar a las subclases	•Es más fácil reimplementar la clase contenedora para utilizar un componente diferente con mínimo impacto en los usuarios que la utilizan
•Las implementaciones de superclase y subclases están ligadas	•Hay menos dependencias de implementación
•Limita la flexibilidad y al final la reutilización	•El comportamiento del sistema dependerá de las interacciones entre objetos en vez de estar definido en una clase

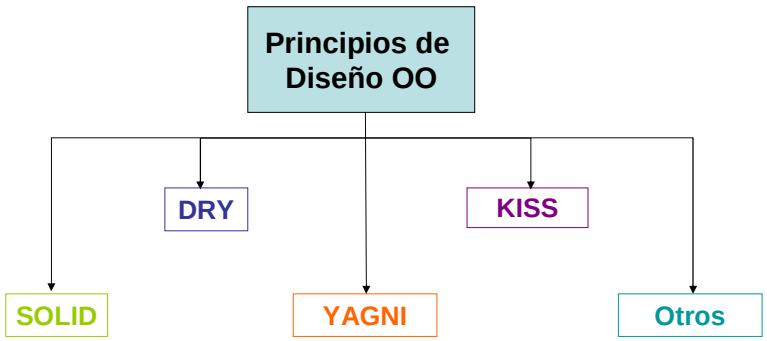
6

Herencia vs Asociación

HERENCIA	ASOCIACIÓN
La herencia no impide a los usuarios manipular la nueva estructura utilizando los métodos de la superclase, aún si estos no son apropiados	Los objetos tienen que respetar las interfaces de los demás objetos
Es **estática**, no puede cambiarse en tiempo de ejecución	Se define **dinámicamente** en tiempo de ejecución

7

Diseño OO



8

Principios de Diseño OO

SOLID - SINGLE RESPONSIBILITY PRINCIPLE

Una clase debería hacer una única cosa y hacerla bien

Sólo porque puedes hacerlo, no significa que debas hacerlo

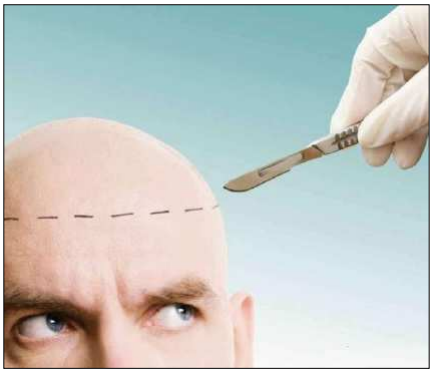
"There should never be more than one reason for a class to change (more than one responsibility)."

Robert C. Martin "Uncle Bob"

Principios de Diseño OO

SOLID - OPEN-CLOSED DESIGN PRINCIPLE

Una clase debería ser abierta para extensión, pero cerrada para modificación



Bertand Meyer, 1988

Una cirugía del cerebro no es necesaria cuando nos ponemos un sombrero

10

Principios de Diseño OO

SOLID - LISKOV SUBSTITUTION PRINCIPLE

Un Subtipo debe ser substituto de un supertipo

Barbara Liskov, 1987

Si tiene el aspecto de un pato, grazna como un pato, pero necesita baterías - Probablemente usted tiene la abstracción equivocada

"The Liskov Substitution Principle states that subtypes must be substitutable for their base types."*

Agile Principles, Patterns, and Practices in C#

Principios de Diseño OO

SOLID - INTERFACE SEGREGATION PRINCIPLE

Evite la interfaz monolítica, reduzca el dolor en el lado del cliente

Adaptar las interfaces a las necesidades de los clientes individuales

"The Interface Segregation Principle states that Clients should not be forced to depend on methods they do not use."

Agile Principles, Patterns, and Practices in C#

Principios de Diseño OO

SOLID - DEPENDENCY INVERSION PRINCIPLE

¿Soldarías una lámpara directamente al cableado eléctrico en una pared?

"Dependency Inversion Principle says that high-level modules should not depend on low-level modules. Both should depend on abstractions."

"Abstractions should not depend on details. Details should depend on abstractions."

Agile Principles, Patterns, and Practices in C#

Principios de Diseño OO

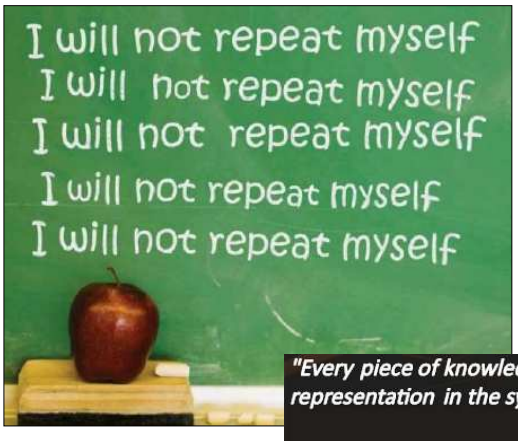
S	Single Responsability Principle
O	Open – Closed Design Principle
L	Liskov Substitution Principle
I	Interface Segregation Principle
D	Dependency Inversion Principle

14

Principios de Diseño OO

DRY – DON'T REPEAT YOURSELF

Evita la duplicación en el código



Andy Hunt and Dave Thomas, 1999

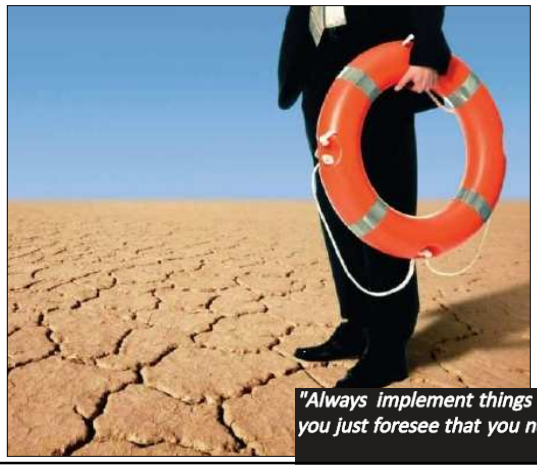
La repetición es la raíz de todos los males del software

"Every piece of knowledge must have a single, unambiguous representation in the system."

The Pragmatic Programmer

Principios de Diseño OO

YAGNI – YOU AIN'T GONNA NEED IT



No desperdicies recursos en lo que podrías necesitar

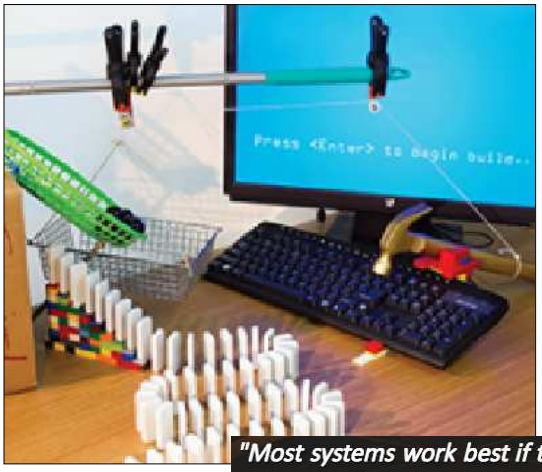
"Always implement things when you actually need them, never when you just foresee that you need them."

Ron Jeffries, XP co-founder

Principios de Diseño OO

KISS – KEEP IT SIMPLE, STUPID

Kelly Johnson, 1960



Si sólo hubiera una forma mas fácil!

"Most systems work best if they are kept simple."

U.S. Navy

Principios de Diseño OO

OTROS PRINCIPIOS

ENCAPSULATES WHAT CHANGES — Oculta los detalles de implementación, ayuda en el mantenimiento

FAVOR COMPOSITION OVER INHERITANCE — Reutilización del código sin costo de inflexibilidad

18

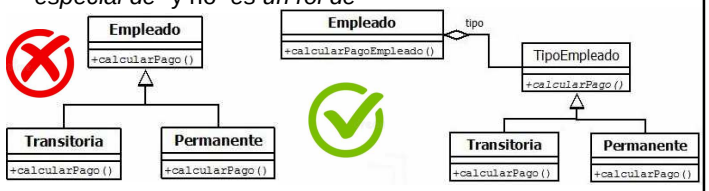
Principios de Diseño OO

FAVOR COMPOSITION OVER INHERITANCE

CONDICIONES

Utilizar herencia **SOLO** cuando se cumplan

- Dentro del dominio del problema las subclases especializan un papel, transacción o dispositivo de la clase base
- Las subclases expresan "*es un caso especial de*" y no "*es un rol de*"



- Una instancia de una subclase nunca necesita convertirse en un objeto de otra subclase
- Las subclases extienden las responsabilidades de su superclase en lugar de sobrescribir y anular las mismas

19

Principios de Diseño OO

OTROS PRINCIPIOS

PROGRAMMING FOR INTERFACE

Ayuda en el mantenimiento, mejora la flexibilidad

VENTAJAS

- El cliente no es consciente de la clase específica del O que usa
- Un O fácilmente puede ser substituido por otro
- Las conexiones de O no tienen que estar *ligadas* a un O de una clase específica, de esta manera se incrementa la flexibilidad
- Disminuye el acoplamiento
- Aumenta la probabilidad de reutilización
- Mejora oportunidades para la Asociación

DESVENTAJA

- Moderado aumento de la complejidad en el diseño

DELEGATION PRINCIPLE

No hagas todas las cosas tu solo, delega

20