DISSO 7 – PATRONES DE DISEÑO

PATRONES

- Los PATRONES son **soluciones** de eficiencia demostrada a **problemas de diseño** comunes.
- Los sistemas que hacen explícitos los patrones usados son más **comprensibles** y **adaptables**.

Tipos de patrones:

- Patrones de **arquitectura** (*frameworks*).
- Patrones de **diseño** (mecanismos) → **plantillas** que demuestran la **estructura** y el **comportamiento** de una sociedad de clases (y objetos).
  - Se representan como **colaboraciones** con aspectos:
    - **Estáticos** → diagramas de clases.
    - **Dinámicos** → diagramas de interacción.
  - **C. Alexander** → cada patrón describe un problema recurrente en un entorno dado y su solución, aplicable un millón de veces sin hacer lo mismo 2 veces.
  - **Paradigma OO** → descripción de clases y objetos que se relacionan para resolver en un determinado contexto un problema de diseño repetido.

REUTILIZACIÓN

- Un diseño debe adecuarse a **requisitos actuales** y anticipar **problemas futuros**.
- Los expertos **no** afrontan cada situación partiendo de **cero**, sino que **reutilizan soluciones** útiles en el pasado.
- Los patrones resuelven problemas concretos de diseño y ayudan a lograr soluciones **flexibles y reutilizables**.

VENTAJAS

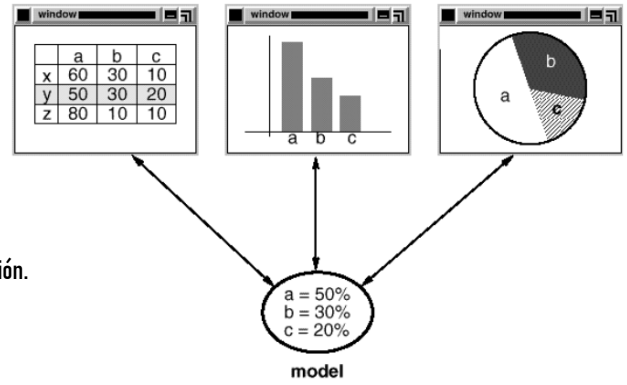
- ✓ Cada patrón nombra, explica y evalúa un **diseño recurrente** en sistemas OO.
- ✓ Permiten lograr **rápidamente** un buen diseño **reutilizando** experiencia previa.
- ✓ Ayudan a **elegir alternativas** que hacen que un sistema sea reutilizable y a evitar las que provocan el efecto contrario.
- ✓ Facilitan la **documentación** y el mantenimiento de sistemas existentes.

ELEMENTOS

- **Nombre** → descripción concisa del problema de diseño.
- **Problema** → contexto de aplicación.
- **Solución** → elementos del diseño y sus relaciones y responsabilidades.
- **Consecuencias** → ventajas e inconvenientes de aplicar el patrón.

MODELO/VISTA/CONTROLADOR

- Tríada de clases para IU en *Smalltalk*:
  - **Modelo** → objeto de aplicación.
  - **Vista** → representación del modelo.
  - **Controlador** → respuesta de la interfaz a la entrada ofrecida por el usuario.
- Las **vistas** y los **modelos** se **desacoplan** usando un protocolo de **suscripción**/notificación.
- Las **vistas** se pueden **anidar**.
- La **respuesta** de una **vista** puede cambiar sin alterar su **representación visual**.



TIPOS DE PATRONES

- Catálogo de patrones GoF →23 patrones de diseño **catalogados** por Gamma et al.
- Clasificación según su **propósito**:
  - De **creación**.
  - **Estructurales**.
  - De **comportamiento**.
- Clasificación según su **ámbito**:
  - De **clase**.
  - De **objeto**.

	De creación	Estructurales	De comportamiento
Clase	**Factory Method**	Adapter	Interpreter **Template Method**
Objeto	**Abstract Factory** Builder Prototype Singleton	**Adapter** **Bridge** **Composite** **Decorator** Facade Flyweight **Proxy**	Chain of Responsibility Command Iterator Mediator Memento **Observer** **State** **Strategy** Visitor

Patrones más sencillos y comunes:

- Abstract Factory
- Factory Method

} De creación

- Adapter
- Composite
- Decorator

} Estructurales

- Observer
- Strategy
- Template Method

} De comportamiento

RESOLUCIÓN DE PROBLEMAS DE DISEÑO

1. Identificación de los objetos necesarios.
2. Determinación del tamaño de los objetos.
3. Programación para interfaces.
4. Reutilización.
5. Diseño para el cambio.

IDENTIFICACIÓN DE LOS OBJETOS NECESARIOS

- Muchos elementos de un diseño proceden del **análisis**.
- Después aparecen clases **sin equivalente en el mundo real**.
  - ▸ Los patrones ayudan a identificar las **abstracciones menos obvias**.

DETERMINACIÓN DE SU TAMAÑO

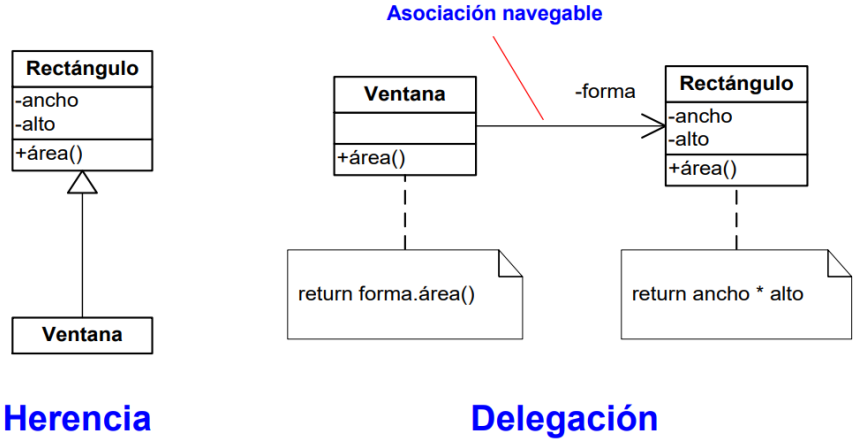
- Los objetos pueden variar notablemente en tamaño y número.
- Algunos patrones de diseño describen formas concretas de **descomponer un objeto en otros más pequeños**.

PROGRAMACIÓN PARA INTERFACES

- Principio de DOO ⟶ **programar para interfaces, no para implementaciones**.
- Se debe **restringir** el **uso de referencias** a clases concretas ⟶ los objetos deben mantener un **conocimiento mutuo limitado**.
- Los **patrones de creación** aseguran que el sistema se diseñe en términos de interfaces.

MECANISMOS DE REUTILIZACIÓN

- HERENCIA ⟶ reutilización de "caja blanca".
  - Se define en tiempo de **compilación**.
  - × **Rompe** el principio de **encapsulación**.
- COMPOSICIÓN ⟶ reutilización de "caja negra".
  - Los enlaces (referencias) entre objetos se establecen en tiempo de **ejecución**.
  - Cualquier objeto puede ser **reemplazado** dinámicamente por otro del **mismo tipo**.
  - ✓ El acceso a objetos a través de interfaces **no rompe** la **encapsulación**.
- Principio de DOO ⟶ **equilibrar el uso de la asociación y la generalización como mecanismos complementarios**.
  - La **herencia** facilita la **construcción**,
    pero la **composición** aporta mayor **flexibilidad**.
- DELEGACIÓN ⟶ modo de asociación que **suple a la herencia** por la cual dos objetos tratan una **petición**: un objeto receptor delega en su **ayudante**.
  - ✓ Facilita **combinación de comportamientos** en tiempo de ejecución.
  - × **Complejidad** añadida.



DISEÑO PARA EL CAMBIO

- La clave para **maximizar la reutilización** es anticipar **nuevos** requisitos y cambios en los requisitos **existentes**.
- Los patrones aseguran que el sistema pueda **evolucionar de forma concreta**.
- Cada patrón permite que algún **aspecto varíe independientemente** de los otros ⟶ **robustez** frente a un tipo de cambio dado.

USO DE LOS PATRONES DE DISEÑO

- Para **seleccionar** un patrón se deben examinar las **causas de rediseño** o decidir qué **aspecto debería ser variable**.
- Los patrones **facilitan el mantenimiento** a cambio de **complicar el diseño** (e incluso **disminuir el rendimiento**).
  - ↳ Se deben aplicar sólo si la flexibilidad proporcionada merece la pena.