Patrones de Diseño: Patrones Estructurales. Tema 4-1: Introducción

Patrones Estructurales

- Los patrones estructurales se centran en como se pueden combinar las clases y los objetos para formar estructuras más grandes y complejas.

- Este tipo de patrones es particularmente útil cuando queremos combinar la funcionalidad de varias bibliotecas de clases que se han desarrollado de forma independiente.

Patrones Estructurales

- Los patrones estructurales de clases generalmente utilizan la herencia para componer interfaces que nos permiten establecer las relaciones entre clases.

- Los patrones estructurales asociados con objetos describen formas de componer los objetos para conseguir nueva funcionalidad. La flexibilidad de la composición de objetos nos da la posibilidad de cambiar la composición en tiempo de ejecución, lo que es imposible con la composición estática de clases.

Clasificación Patrones Estructurales

- **Adapter**: Convierte la interfaz que ofrece una clase en otra esperada por otra clase. Sirve de intermediario entre dos clases.

- **Bridge**: Desacopla una abstracción de su implementación y les permite variar independientemente. Divide un componente complejo en dos jerarquías relacionadas, la abstracción funcional y la implementación interna. Esto hace que sea más fácil cambiar cualquier aspecto del componente.

Clasificación Patrones Estructurales

- **Composite**: Permite gestionar objetos complejos e individuales de forma uniforme. Desarrolla una forma flexible de crear jerarquías en estructura de árbol de una complejidad arbitraria, permitiendo a la vez que todos los elementos de la estructura funcionen con una interfaz uniforme.

- **Decorator**: Extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes. Proporciona una forma flexible de introducir o eliminar funcionalidad de un componente sin modificar su apariencia externa o su función.

Clasificación Patrones Estructurales

- **Facade**: Simplifica los accesos a un conjunto de objetos relacionados proporcionando una interfaz de comunicación.
- **Flyweight**: Reduce el número de objetos detallados de muy bajo nivel en un sistema usando la compartición de objetos de forma eficiente.
- **Proxy**: Proporciona un objeto (representante) con el que controlamos el acceso a otro objeto.