Exposición Patrones de diseño

Escoger uno de los patrones de diseño que hay en la lista de abajo. Buscar información en internet y crear una presentación, para su posterior exposición, donde se explique en qué consiste el patrón, qué problemas soluciona, ventajas y desventajas, relación con otros patrones que abordan problemas parecidos, ejemplo de implementación del patrón en código y diagrama de clase genérico del patrón y específico de dicha implementación. Duración de la exposición 15 minutos mínimo.

Singleton: Este patrón garantiza que una clase tenga una sola instancia y proporciona un punto de acceso global a esa instancia. Es útil cuando solo se necesita una instancia de una clase en toda la aplicación.

Service Locator: Este patrón se utiliza para localizar los servicios (como objetos, componentes o recursos) en una aplicación. Proporciona una forma centralizada para acceder a estos servicios sin acoplar el código que los usa con la implementación específica de cada servicio.

Dependency Injection: Este patrón se utiliza para desacoplar componentes en un sistema de software, lo que facilita la creación de código modular y mantenible.

Factory Method: Define una interfaz para crear un objeto, pero permite a las subclases alterar el tipo de objetos que se crearán. Es útil cuando una clase no puede anticipar la clase de objetos que debe crear.

Builder: Se utiliza para construir objetos complejos paso a paso. El patrón permite producir diferentes tipos y representaciones de un objeto utilizando el mismo proceso de construcción.

Observer: Define una dependencia uno a muchos entre objetos de modo que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente. Es útil cuando un objeto necesita notificar a otros objetos sobre cambios en su estado.

Strategy: Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Este patrón permite que el algoritmo varíe independientemente de los clientes que lo utilizan.

Decorator: Permite añadir comportamiento a objetos individuales, de forma dinámica, sin afectar a otros objetos del mismo tipo. Es útil cuando se necesitan funcionalidades adicionales que pueden ser agregadas y eliminadas fácilmente.

Composite: Se utiliza para tratar a las colecciones de objetos de manera similar a los objetos individuales. Este patrón permite a los clientes tratar tanto a objetos individuales como a colecciones de objetos de manera uniforme, ya que ambos comparten una interfaz común.

Command: Encapsula una solicitud como un objeto, permitiendo parametrizar clientes con operaciones, encolar solicitudes y soportar operaciones reversibles.

Entrega

Hay que entregar, en la tarea correspondiente en la moodle, la presentación en formato pdf y el código de la demo dentro de un zip.