Clase 13/06 - 22/06 Patrones de Diseño

Patrones Creacionales

- Soluciones comúnes a problemas relacionados con creación de objetos en SW Design.
- Proporcionan formas flexibles y reutilizables de crear y configurar objetos.

Singleton

- Garantizar que una clase solo tenga una única instancia en todo el programa, proporcionando un punto de acceso global a esa instancia.

Contras

- Viola SRP
- Enmascara malos diseños:
  - Los cokmponentes del programa saben demasiado los unos sobre los otros
  - Produce acoplamiento
  - Complica automatización de tests.

Factory Method

- Proporciona una interfaz para crear objetos en una superclase y permite a las subclases alterar el tipo de objetos que se generan.

Usos

- Cuando se trabajan con jerarquias de clses y se necesitan crear obj polimórficos

Builder

- Permite construir objetos complejos paso a paso, permitiendo también producir istintos tipos y representaciones de un objeto

Usos

- Es especialmente útil cuando existe un objeto complejo que requiere una inicialización laboriosa, paso a paso,de muchos campos y objetos anidados
- El patrón Builder facilita la modificación y apliación del proceso de construcción
- Se puede aplicar cuando la construcción de varias representaciones de un producto requiera de pasos similares qeu solo varían en detalles.

Contras

- Complejidad del código aumenta, el patrón exige la creación de clases nuevas.

Prototype

- Se utiliza para crear nuevos objetos a través de la clonación de un objeto existente.

Estructura

- Debe existir una interfaz “Prototype” que declara los métodos de clonación, por ejemplo: clonar()
- Existe una clase que implementa Prototype que llamaremos ConcretePrototype
- La clase ConcretePrototype implementa el método clonar()
- Si el método clonar() copia las referencias de la clase “ConcretePrototype”el clonado se denomina como superficial
- De lo contrario, si copia los objetos de las referencias estamos ante una clonación profunda.

Contras

- Puede haber complicaciones con referencias circulares si emplea clonación profunda.

Adapter

- Utilizado cuando quiero usar una clase existente cuya interfaz no sea compatible con otra parte del código.

Contra

- Aumenta la complejidad general del código

Bridge

- Patron de diseño estructural que permite **dividir una clase/grupo de clase en dos jerarquias separadas** que pueden desarrollarse independientemente la una de la otra.

Contra

- Puede ser que el código se complique si aplicas el patrón a una clase muy cohesionada.

Adapter vs Bridge

Propósito

- **Adapter**: Permitir que dos interfaces incompatibles trabajen junts
- **Bridge**: Desacoplar una abstracción de su implementación para que puedan variar de forma independiente.

Intención

- Adapter: Resolver problemas de incompatibilidad entre interfaces ya existentes
- Bridge: Planificado de antemano

Estructura

- Adapter: En gral, el patrón Adapter envuelve la clase adaptada y expone una interfaz que el cliente puede usar. No hay una separación clara entre abstracción y separación
- Bridge: Tiene una estructura más compleja con una clara separación entre la abstracción y la separación.

Decorator

- Utilizar este patrón cuando necesites asignar funcionalidades adicionales a objetos durante el tiempo de ejecución sin descomponer el código que utiliza el sistema

Contras

- Resutlta dificil eliminar un wrapper especifico de la pila de wrappers

Facade

- Proporciona una interfaz unificada a un conjunto de interfaces en un subsistema. Define una interfaz de nivel superior que facilita el uso del subsistema.

Proxy

Chain of Responsability

- Genera una abstracción y un pipeline de despliegue de tareas que se pueden ejecutar de forma dinámica. Es una forma de conectar los objetos, con cada emisor conociendo a su padre.

Command

- Transforma una solicitud en un objeto independiente con toda la información sobre la solicitud.

Observer

- Define una dependencia 1 a N con objetos
    - Cuando un objeto cambia de estado, notifica a todos los dependientes de que hubo un cambio, para que ellos se adapten

Mediator

- Se usa para reducir la comunicación compleja entre objetos estrechamente relacionados. En lugar de que los objetos se comuniquen directamente entre sí, estos objetos interactúan a través del **mediador**.
    - Al tener un conjunto de objetos etrechamente relacionados y que necesitan comunicarse entre sí de maneras complejas. En lugar de tener un número creciente de conexiones entre cada par de objetos, solo necesitas conectar c/objeto con el mediador.

Observer VS Mediator

- **Uso Común**:

Memento

- Captura el estado interno de un objeto en un punto en el tiempo, de modo que el objeto pueda ser restaurado más tarde.

Strategy

- Permite seleccionar un algoritmo o estrategia en tiempo de ejecución. En lugar de implementar un único algrotimo directamente dentro de una clase, el patrón Strategy utiliza interfaces para hacer que un conjunto de algoritmos sean intercambiables.

Visitor

- Permite separar algoritmos de los objetos sobre los que operan. Esto puede ser útil cuando necesitan realizar operaciones sobre estos objetos sin alterar sus estados.

State, Strategy y Visitor

- **Uso Comun**
  - State: Cuando el comportamiento del objeto debe cambiar dinámicamente en función de su estado
  - Strategy: Cuando se necesita elegir entre varias implementaciones de un algoritmo o comportamiento en tiempo de ejecución
  -