

PATRONES CREACIONALES

Integrantes del Grupo 1

- | | |
|-----------------------------------|---------------|
| 1. Carlos René palma Ramírez | 0907-23-900 |
| 2.Timothy Gerald Palma Perez | 0907-20-6162 |
| 3.Héctor de Jesús de Paz Arbizú | 0907-23-12173 |
| 4.Mario Josué Tobar Herrera | 0907 22 9109 |
| 5.Hugo Alfredo Sagastume Coronado | 0907 19 21049 |
| 6.Gencer Uriel Cortéz Ramírez | 0907-22-10331 |



1

**Factory
Method**

2

**Abstract
Factory**

3

Builder

4

Prototype

5

Singleton



FACTORY METHOD

es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

Ventajas

- Desacoplamiento: Elimina la necesidad de instanciar directamente clases concretas en el código cliente, lo que reduce el acoplamiento entre las clases.
- Extensibilidad: Es fácil agregar nuevas clases de productos sin modificar el código existente.
- Control de creación: Las subclases pueden controlar cómo y qué objetos se crean.

Desventajas

- Más Clases y Código: Implementar el patrón Factory Method puede aumentar el número de clases y líneas de código en un proyecto.
- Sobrecarga de Clases: En escenarios simples, la creación de una clase de fábrica separada puede ser excesiva y hacer que el diseño parezca innecesariamente complicado.

ABSTRACT FACTORY

es un patrón de diseño creacional que proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas. A diferencia del Factory Method, que crea objetos de un solo tipo, el Abstract Factory se usa cuando se necesita crear varios tipos de objetos que están relacionados entre sí.

Ventajas

- **Desacoplamiento del Cliente de las Clases Concretas:** Permite que el cliente esté desacoplado de las implementaciones específicas de los productos, facilitando la modificación y extensión del sistema sin cambiar el código del cliente.
- **Facilita el Cambio de Implementaciones:** Esto facilita la adaptación a nuevas necesidades o la integración de nuevas tecnologías sin afectar el resto del sistema.

Desventajas

- **Complejidad Adicional:** Introduce complejidad adicional en el diseño del sistema debido a la necesidad de crear varias clases abstractas y concretas para cada familia de productos, lo que puede hacer que el código sea más difícil de entender.
- **Dificultad en la Extensión:** Aunque facilita la extensión en términos de nuevas familias de productos, puede ser menos flexible si se necesita modificar la estructura de las familias de productos existentes.
- **Sobreabundancia de Clases:** Puede llevar a una proliferación de clases, ya que se crean diferentes clases concretas para cada tipo de producto y sus variantes, lo que puede hacer que el diseño sea más complejo y difícil de gestionar.

BUILDER

El patrón Builder es un patrón de diseño creacional que permite construir un objeto complejo mediante una serie de pasos simples. El patrón separa la construcción de un objeto de su representación final, de modo que el mismo proceso de construcción puede crear diferentes representaciones.

Ventajas

- Flexibilidad: El patrón Builder nos permite construir objetos complejos con diferentes configuraciones sin tener que modificar el código existente.
- Legibilidad: Al separar el proceso de construcción en pasos más pequeños, el código se vuelve más legible y fácil de entender.
- Mantenibilidad: El patrón Builder promueve el principio de responsabilidad única al separar la construcción del objeto de su representación final.

Desventajas

- Complejidad adicional: Al utilizar el patrón Builder, se añade una capa adicional de complejidad al código. Se requiere la implementación de múltiples clases (Builder, ConcreteBuilder, Director).
- Acoplamiento entre el Builder y el producto: En algunos casos, el Builder puede estar acoplado estrechamente al producto que se está construyendo.

PROTOTYPE

basa su funcionalidad en la clonación de objetos, estos nuevos objetos son creados mediante un pool de prototipos elaborados previamente y almacenados. Este patrón es especialmente útil cuando necesitamos crear objetos basados en otros ya existentes o cuando se necesita la creación de estructuras de objetos muy grandes, este patrón nos ayuda también a ocultar la estrategia utilizada para clonar un objeto.

Ventajas

- Creación eficiente de objetos: El patrón Prototype permite crear nuevos objetos clonando un prototipo existente en lugar de instanciar una clase desde cero.
- Simplificación del código: El patrón Prototype puede simplificar el código al evitar la necesidad de crear múltiples subclases o realizar configuraciones complicadas.

Desventajas

- Complejidad de clonación: Al clonar objetos, debes tener cuidado con la clonación profunda (copia de todas las propiedades y referencias internas) y la clonación superficial (copia solo de las propiedades superficiales).
- Problemas con la mutabilidad: Si el prototipo es mutable y se realizan cambios en una instancia clonada, los cambios pueden afectar a otras instancias clonadas y al prototipo original.

SINGLETON

El patrón de diseño Singleton (soltero) recibe su nombre debido a que sólo se puede tener una única instancia para toda la aplicación de una determinada clase, esto se logra restringiendo la libre creación de instancias de esta clase mediante el operador new e imponiendo un constructor privado y un método estático para poder obtener la instancia.

Ventajas

- Control de Instancias Únicas: Garantiza que solo haya una instancia de la clase en todo el sistema, lo que puede ser útil para gestionar recursos compartidos o configurar una única fuente de datos.
- Acceso Global: Proporciona un punto de acceso global a la instancia única, lo que facilita la comunicación entre diferentes partes del sistema que necesitan acceder a la misma instancia.
- Reducción de Carga de Recursos: Al evitar la creación de múltiples instancias, se pueden reducir los costos asociados con la inicialización y gestión de objetos.

Desventajas

- Problemas con las Pruebas Unitarias: Puede dificultar las pruebas unitarias porque la instancia única puede estar involucrada en diferentes pruebas, lo que puede causar efectos secundarios indeseados y acoplar los tests.
- Acoplamiento Global: Puede llevar a un acoplamiento global en el sistema, donde muchas partes del código dependen de la misma instancia, lo que puede hacer que el código sea más difícil de entender y mantener.
- Falta de Flexibilidad: El patrón Singleton puede hacer que el sistema sea menos flexible y extensible porque la clase Singleton está estrictamente controlada y puede ser difícil de modificar sin afectar otras partes del sistema.