Manuel Alonso Araniva Valle

Patrones de diseño

¿Qué son los patrones de diseño?

Los patrones de diseño son soluciones habituales a problemas comunes en el diseño de software.

Ventajas de los patrones

1. Estandarizar el lenguaje entre programadores
2. Evitar perder tiempo en soluciones a problemas ya resueltos o conocidos
3. Crear código reusable.

Tipo de patrones de diseño

1. **Creacionales:** su objetivo es resolver los problemas de creación de instancias. Estos ayudan a delegar la responsabilidad de creación de objetos en situaciones necesarias.
2. **Estructurales:** se ocupa para resolver problemas sobre la estructura de las clases, es decir se enfoca en cómo las clases y objetos se componen para formar estructuras mayores.
3. **Comportamiento:** ayuda a resolver problemas relacionados con el comportamiento de la aplicación respecto a la interacción y responsabilidad entre objetos y clases.

Patrones de diseño

Singleton: es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

Para implementar el patrón Singleton se necesita lo siguiente:

- Hacer privado el constructor por defecto para evitar que otros objetivos utilicen el operador new con la clase Singleton
- Crear un método de creación estático en el que actué como constructor.

```java
package refactoring_guru.singleton.example.non_thread_safe;

public final class Singleton {
    private static Singleton instance;
    public String value;

    private Singleton(String value) {
        // The following code emulates slow initialization.
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        this.value = value;
    }

    public static Singleton getInstance(String value) {
        if (instance == null) {
            instance = new Singleton(value);
        }
        return instance;
    }
}
```

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.

Características del Factory Method:

1. **Desacoplamiento**: Desacopla la creación del objeto de su uso, lo que facilita la extensión y mantenimiento del código.
2. **Extensibilidad**: Nuevas clases pueden ser añadidas sin modificar el código existente, simplemente extendiendo la clase que implementa el método de fábrica.
3. **Polimorfismo**: Permite el uso del polimorfismo para manipular objetos de diferentes tipos a través de una interfaz común

Builder: es un patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

Características del Patrón Builder:

1. **Construcción paso a paso**: Permite construir un objeto paso a paso, proporcionando métodos específicos para cada paso de la construcción.
2. **Inmutabilidad**: Puede ayudar a crear objetos inmutables, ya que todos los campos del objeto se pueden inicializar a través del Builder.
3. **Fluidez**: A menudo implementado con un enfoque fluido, permitiendo encadenar llamadas a métodos (.setX().setY()...)

Prototype: es un patrón de diseño creacional que se utiliza para crear nuevos objetos copiando o clonando una instancia existente en lugar de crear un nuevo objeto desde cero. Este patrón es especialmente útil cuando la creación de un objeto es costosa o compleja, y queremos evitar ese costo duplicando un objeto existente

Características del Patrón Prototype:

1. **Clonación**: Permite crear nuevos objetos copiando/clonando objetos existentes.
2. **Flexibilidad**: Puede ser utilizado para evitar la creación de subclases redundantes, ya que un prototipo existente puede ser modificado antes de ser clonado.
3. **Reducción de costos**: Ideal cuando la creación de un objeto es costosa en términos de recursos o tiempo

Adapter: es un patrón de diseño estructural que permite que clases con interfaces incompatibles trabajen juntas. Actúa como un puente entre dos interfaces incompatibles, convirtiendo la interfaz de una clase en otra que la clase cliente espera

Características del Patrón Adapter:

1. **Interfaz de compatibilidad**: Permite que clases con interfaces incompatibles colaboren entre sí.
2. **Reutilización de código**: Facilita la reutilización de clases existentes que no se pueden modificar directamente.
3. **Flexibilidad**: Se puede aplicar tanto a clases (herencia) como a objetos (composición)