PATRONES DE DISEÑO

ANA MARIA MORENO CASADIEGO 1152073
YEINER DANIEL ANAYA 1152086
MOISES OSORIO 1152082

PRESENTADO
RODRÍGUEZ TENJO JUDITH DEL PILAR

UNIVERSIDAD FRANCISCO DE PAULA SANTANDER
1155705- ANALISIS Y DISEÑO DE SISTEMAS
CÚCUTA

REUTILIZACIÓN DEL SOFTWARE

Patrones de Diseño

El diseño OO es difícil y el diseño de software orientado a objetos reutilizables lo es aún más.

• Los diseñadores expertos no resuelven los problemas desde sus principios; reutilizan soluciones que han funcionado en el pasado.

– Se encuentran patrones de clases y objetos de comunicación recurrentes en muchos sistemas orientados a objetos.

– Estos patrones resuelven problemas de diseño específicos y hacen el diseño flexible y reusable.

patrón

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y describe también el núcleo de la solución al problema, de forma que puede utilizarse un millón de veces sin tener que hacer dos veces lo mismo

Los patrones de diseño ayudan a capturar y reutilizar la experiencia y conocimiento de diseñadores expertos. Proporcionan un vocabulario y entendimiento comunes de los problemas de diseño y sus soluciones. No son diseños terminados que puedan transformarse directamente en código, sino descripciones o plantillas sobre cómo resolver un problema, que pueden usarse en muchas situaciones diferentes (Gamma et al., 1995).

Cada patrón de diseño tiene esencialmente cuatro elementos (Gamma et al., 1995):

1. El nombre del patrón, que lo identifica y describe el problema, sus soluciones y consecuencias de manera sucinta.
2. El problema, que describe cuándo aplicar el patrón, explica el problema y su contexto.
3. La solución, que describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboraciones.
4. Las consecuencias, que son los resultados y compromisos de aplicar el patrón.

Clasificación La GoF clasificó 23 patrones de diseño según dos criterios (Gamma et al., 1995):

Propósito, que refleja lo que hace el patrón:

- Creacionales: Conciernen al proceso de creación de objetos.
- Estructurales: Tratan con la composición de clases u objetos.
- De Comportamiento: Caracterizan las formas en que las clases u objetos interactúan y distribuyen la responsabilidad.

Ámbito, que especifica si el patrón se aplica principalmente a clases u objetos:

- Clase: Tratan con relaciones entre clases y sus subclases. Estas relaciones se establecen a través de la herencia, por lo que son estáticas en tiempo de compilación.
- Objeto: Tratan con relaciones entre objetos, que pueden cambiarse en tiempo de ejecución y son más dinámicas.

Patrones creacionales Abstraen el proceso de creación de instancias de objeto. Ayudan a hacer un sistema independiente de cómo se crean, componen y representan sus objetos. Incrementan la flexibilidad en términos de qué se crea, quién lo crea, cómo se crea y cuándo. Permiten encapsular el conocimiento sobre qué clases concretas utiliza el sistema. Hay dos ideas recurrentes: Encapsular el

conocimiento sobre qué clases concretas utiliza el sistema, y ocultar cómo se crean y combinan las instancias de estas clases (Gamma et al., 1995). Algunos ejemplos:

- Singleton: Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella (Gamma et al., 1995). Es útil para gestionar recursos compartidos, como conexiones a bases de datos, sistemas de archivos, etc.
- Factory Method: Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos (Gamma et al., 1995). Es útil para encapsular la creación de objetos y hacer los sistemas más flexibles y extensibles.
- Abstract Factory: Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas (Gamma et al., 1995). Es útil para sistemas que necesitan ser independientes de cómo se crean sus objetos.

Patrones estructurales

Se ocupan de cómo se combinan las clases y los objetos para formar estructuras más grandes. Los patrones de clase estructurales usan la herencia para componer interfaces o implementaciones, mientras que los patrones de objeto describen formas de componer objetos para obtener nuevas funcionalidades. La flexibilidad obtenida al componer objetos viene de la capacidad de cambiar la composición en tiempo de ejecución, lo cual es imposible con la composición estática (herencia de clases) (Gamma et al., 1995). Algunos ejemplos:

- Adapter: Convierte la interfaz de una clase en otra interfaz que los clientes esperan. Permite que clases con interfaces incompatibles trabajen juntas (Gamma et al., 1995). Es útil para integrar clases de terceros o antiguas en un sistema nuevo.
- Composite: Compone objetos en estructuras de árbol para representar jerarquías parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los

compuestos (Gamma et al., 1995). Es útil para representar jerarquías recursivas o cuando se quiere que el código cliente pueda ignorar la diferencia entre objetos simples y compuestos.

- Facade: Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar (Gamma et al., 1995). Es útil para proporcionar una interfaz simplificada a un subsistema complejo o a una colección de subsistemas.

Patrones de comportamiento Se ocupan de los algoritmos y de la asignación de responsabilidades entre objetos. Los patrones de comportamiento de clases usan la herencia para distribuir el comportamiento entre clases, mientras que los patrones de comportamiento de objetos usan la composición de objetos en lugar de la herencia. Algunos describen cómo grupos de objetos igual colaboran para realizar una tarea que ningún objeto individual puede llevar a cabo por sí solo. Otros patrones se encargan de encapsular el control sobre un algoritmo o un comportamiento que depende del estado (Gamma et al., 1995). Algunos ejemplos:

- Observer: Define una dependencia de uno a muchos entre objetos, de manera que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente (Gamma et al., 1995). Es útil para mantener consistencia entre objetos relacionados sin acoplarlos fuertemente.
- Strategy: Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permite que el algoritmo varíe independientemente de los clientes que lo usan (Gamma et al., 1995). Es útil cuando hay muchas clases relacionadas que sólo difieren en su comportamiento, o cuando se necesitan diferentes variantes de un algoritmo.
- Iterator: Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna (Gamma et al., 1995). Es útil para soportar múltiples recorridos sobre una estructura de datos o cuando se quiere una interfaz uniforme para recorrer diferentes estructuras.

Patrones fundamentales Además de los 23 patrones de la GoF, el documento cubre algunos patrones fundamentales adicionales:

- Delegation: Cuando se quiere extender y reutilizar la funcionalidad de una clase sin usar herencia, delegando parte del comportamiento en otra clase en lugar de heredarlo.
- Interface: Define un comportamiento independiente de dónde vaya a ser utilizado, proporcionando una interfaz común que las clases pueden implementar para ciertos propósitos y facilitando el desarrollo de clases utilitarias genéricas.
- Marker Interface: Sirve para marcar o indicar atributos semánticos de una clase sin necesidad de implementar ningún método, permitiendo a código genérico tratar a las clases marcadas de manera especial.

Referencias Bibliográficas

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). A Pattern Language: Towns, Buildings, Construction. Oxford University Press.

- Enlace: https://www.amazon.com/Pattern-Language-Buildings-Construction-Environmental/dp/0195019199

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

- Enlace: https://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612