Patrones de diseño de software

Rodríguez Galindo Juan Felipe
Código - 20181020158
juafrodriguezg@correo.udistrital.edu.co
Profesor: Malaver Jorge Armando

***Resumen*—En el actual documento se revisaran los diferentes patrones aplicados en el diseño de software sus definiciones, donde y como se pueden aplicar en los programas que realicemos. También se enumerarán algunos ejemplos de donde podemos utilizarlos y realizar una pequeñas conclusiones definiendo el punto de vista del autor.**

***Palabras clave*—Patrones, Software, Diseño, Modelos de programación, Java.**

I. Introducción

Este articulo presenta el uso, aplicación y manejo de patrones dentro del desarrollo de software. Se fundamenta en explicar al lector el conocimiento básico de los patrones y el como nosotros dentro de nuestra carrera podemos aplicar estos conocimientos para fabricar software de calidad.

La aplicación de patrones *Factory, Factory Method*, entre otros; en las circunstancias correctas agregan al software la escalabilidad y mantenibilidad que un software de calidad posee, por esto el buen entendimiento de los patrones es fundamental para aplicar en el desarrollo de software.

Por lo que el articulo pretende mostrar al lector, que es un patrón de diseño, algunas de sus características y los casos en los que pueden ser empleados de la mejor forma.

El documento se dividirá en tres grandes etapas o secciones, en la primera se le dará una introducción a que es un patrón de diseño, luego se verán algunos de los tipos más reconocidos y su aplicación, ya para la sección final se concluirá con la percepción del autor sobre la temática.

II. Patrones de diseño

Los patrones de diseño son una serie o conjunto de técnicas, que buscan solucionar problemas comunes dentro del desarrollo de software de una forma general, es decir, que se basa en una estructura de código reutilizable y que se pueda aplicar en problemas en los cuales su planteamiento es muy similar el uno con el otro.

Todos estos patrones siguen también una serie de objetivos los cuales se pueden sintetizar en estandarizar el código generado, evitar çrear la rueda de nuevo", ser una basta serie de opciones reusables en distintos proyectos informáticos. De igual como hay un objetivo que se debe cumplir, también existe una serie de aspectos que no se pretenden realizar estas son: imponer o ser estrictos en que solo hay una forma de realizar las cosas y no pretende bloquear o mitigar la creatividad del desarrollador.

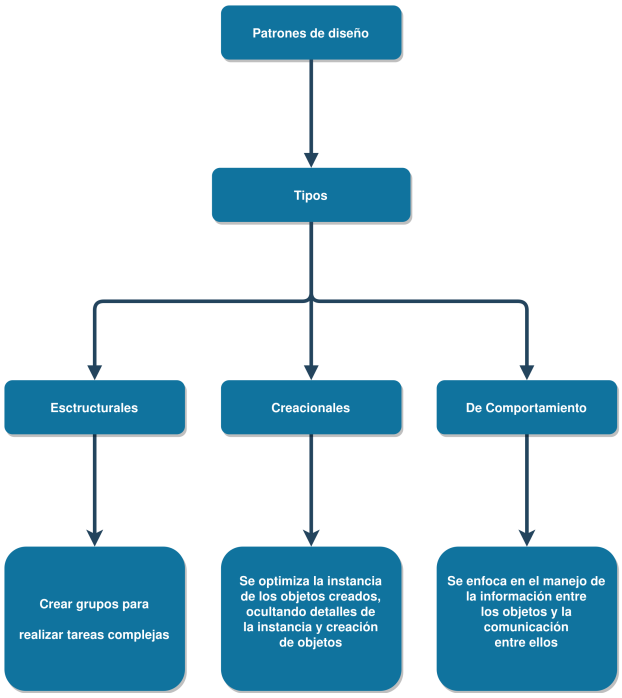

Figura 1. Mapa conceptual modelos y tipos.

También los patrones de diseño se pueden clasificar según el alcance de los mismos, pudiendo ser de clase (los cuales se basan en la herencia de las clases) o de objetos (los cuales se fundamentan en la utilización dinámica de los objetos).

Se puede sintetizar que un patrón de diseño es todo aquel que contiene los cuatro elementos que lo componen:

1. **Nombre del patrón:** Con una o dos palabras logra describir el problema, consecuencias y solución que aporta el patrón.
2. **Problema:** Explica el problema planteado que logra solucionar junto con el debido contexto para su aplicación.

El presente documento corresponde al articulo realizado para la materia modelos de programación 1 de ingeniería de sistemas presentado en la Universidad Distrital FJC durante el periodo 2021-1.

3. **Solución:** Describe todos los elementos que inciden dentro del patrón tales como la estructura del diseño, las relaciones, responsabilidades y colaboraciones. Lo que no realiza es la implementación especifica del patrón, en cambio se dedica a proveer una plantilla capaz de ser una solución abstracta y aplicable a un gran número de problemas similares.
4. **Consecuencia:** se basa en todos aquellos resultados que se pueda obtener posteriormente a la aplicación del patrón. Se ve también como una forma de poder reconocer si es la mejor opción para utilizar según los resultados o es mejor otro; en si se puede considerar como una lista de ventajas y desventajas de aplicar el patrón.

Aunque existen a su vez otros tipos de categorías diferentes de los patrones ya vistos donde podemos encontrar, por ejemplo:

- **Antipatrones :** Tratan de forma contraria que los patrones de diseño encontrar los problemas o trampas que se puedan presentar en los diseños de software, con el objetivo de que no se vuelvan a cometer o no se cometan dentro del diseño del software.
- **Patrones de Programación (Idions) :** Son patrones de bajo nivel acerca de un lenguaje especifico de programación, en los cuales se describe el como debe ser utilizada cierto modelo. Están a su vez en un nivel menor que los patrones de diseño.
- **Patrones de Análisis:** Son un compendio de normas que le permiten al desarrollador generar el modelado de un sistema de información de forma satisfactoria.
- **Patrones de Asignación de responsabilidades:** Describen los principios de la asignación de responsabilidades a cierto objeto de tal forma que su representación sea expresada como un patrón de diseño.
- **Patrones de Arquitectura:** Se trata de como se debe descomponer, conectar y generar relaciones entre sistemas, cubriendo conceptos tales como: filtros, tuberías y niveles. Se considera que posee un nivel de abstracción mucho más alto que el de los patrones de diseño.
- **Patrones organizacionales:** Se encargan de describir el como se deberían organizar los grupos humanos, que generalmente son asociados con el software.
- **Otros patrones de software:** Patrones de interfaz gráfica, de organización del código, de robustez de código entre otras.

III. Tipos de patrones

[1] Como lo evidenciamos anteriormente existen una serie de tipologías de patrones las cuales también nos sirven para escoger el mejor patrón orientado a nuestro caso de uso o problema en especifico, dentro de estos existen tres categorías.

III-A. Estructurales

Expresan una organización arquitectónica básica para la construcción de un sistema de software, incidiendo directamente en la forma en que son agrupados tanto los objetos como las clases para poder generar estructuras complejas o mas grandes. Usualmente este tipo de patrones estructurales orientados a las clases utilizan la herencia para dar o crear nuevas funcionalidades.

Patrones de creación de clase:

- **Adaptador de Clases :** Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.

Patrones de creación de objetos:

- **Adaptador de Clases**
- **Puente :** Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.
- **Compuesto :** Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.
- **Decorador :** Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
- **Fachada :** Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema.Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar.
- **Flyweigth :** Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.
- **Proxy :** Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

III-B. De Creación

Estos abstraen la forma en la que son creados los objetos, de tal manera que le permiten tratar las clases de forma mucho más genérica, separando la decisión de que clases crear o como crearlas. Según de como se desee implementar el patrón se hablara de patrones de clase (en el cual utiliza la herencia para determinar la creación de las instancias, es decir en los constructores de las clases) o bien también se podría recurrir a llevar el modelo a patrones de objeto (donde se vuelve relevante la modificación de los métodos). Los patrones de creación otorgan diversas formas de eliminar las referencias explicitas a clases concretas del código donde deben ser utilizadas.

Patrones de creación de clase:

- **Factoría Abstracta :** Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
- **Constructor :** Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

Patrones de creación de objetos:

- **Método de factoría :** 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.

- **Prototipo :** Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo.
- **Singleton :** Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.
- **Conjunto de objetos**

III-C. De comportamiento

Estos patrones están estrechamente ligados con los algoritmos y la asignación de responsabilidades entre objetos, como característica fundamental de este tipo de patrones no describen únicamente patrones de clases y objetos, ya que estos también se encargan de especificar como se comunican entre sí, también por caracterizar complicados flujos de control que son complejos al momento de darle seguimiento o imaginar el tiempo de ejecución.

Principales patrones de comportamiento:

- **Cadena de responsabilidad :** Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.
- **Comando :** Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.
- **Interprete :** Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.
- **Mediador :** Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.
- **Memento :** Representa y externaliza el estado interno de un objeto sin violar la encapsulación,de forma que éste puede volver a dicho estado más tarde.
- **Observador :** Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.
- **Estrategia :** Define una familia de algoritmos,encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.
- **Plantilla:** Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.
- **Visitante :** Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

IV. CONCLUSIONES

Se logra evidenciar que para cada tipo de problema existe un patrón que puede llegar a solucionar el problema que se tenga, también se logra evidenciar que el uso de patrones es una muy buena practica de programación siempre que se considere todos los aspectos que se deban cumplir en el modelado del sistema. También se puede concluir que no siempre es necesario utilizar este tipo de patrones dentro de nuestros proyectos, esto lo expresa el autor al momento de aclarar que no se trata de cerrar o limitar al programador al estricto uso de los patrones, sino que de hecho se puede innovar y crear nuevos patrones para nuevos problemas.

REFERENCIAS

[1] A. Daza , *Patrones de diseño, un acercamiento - generalidades*, #version 1. Bogotá D.C., Colombia: 2017.
[2] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns. Elements of Reusable Object-OrientedSoftware*, 3rd ed. Pearson Education India, 1995.
[3] Patrones. http://artemisa.unicauca.edu.co/~alerios/publicaciones/patrones.pdf. Recuperado el 14 de Abril de 2021.
[4] Patrones de Diseño, Diseño de Software Orientado a Objetos-Joaquin Garcia. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php. Recuperado el 14 de Abril de 2021.
[5] Patrones de diseño. http://es.kioskea.net/contents/genie-logiciel/design-patterns.php3. Recuperado el 14 de Abril de 2021.
[6] Introducción al diseño con patrones. Miguel Lagos http://www.elrincondelprogramador.com/default.asp?pag=articulos/leer.asp&id=29. Recuperado el 15 de Abril de 2021.
[7] Object Oriented Desing "Software Desing Principles and Design Patters. http://www.oodesing.com/. Recuperado el 15 de Abril de 2021.
[8] Patrones en la ingeniería de software. Jesús Cumare. https://ingsoftwarei2014.wordpress.com/2014/07/03/patrones-en-la-ingenieria-de-software-5/. Recuperado el 15 de Abril de 2021.
[9] Patrones GoF java. Alejandro Daza. https://github.com/apdaza/patrones-gof-java. Recuperado el 15 de Abril de 2021.
[10] Patrones. Francisco José Garcia. https://webcache.googleusercontent.com/search?q=cache:6i6hZxTLJoAJ:https://repositorio.grial.eu/bitstream/grial/356/1/patrones1.pdf+&cd=4&hl=es&ct=clnk&gl=co. Recuperado el 16 de Abril de 2021.
[11] Patrones. David Hernández Tajada. https://web.archive.org/web/20040831083023/http://webs.teleprogramadores.com/patrones/#3.1.-%20%20Adaptador%20-%20Class%20Adapter%20-%20Object%20Adapter%20-%20Wrapper%20. Recuperado el 16 de Abril de 2021.
[12] Patrones de diseño GoF. Max, Mi granito deJava. http://migranitodejava.blogspot.com/search/label/Introduccion%20a%20Patrones. Recuperado el 16 de Abril de 2021.