Tecnológico Nacional de México

Instituto Tecnológico de Tijuana

SEMESTRE:

Agosto-Diciembre 2023

CARRERA:

Ingeniería en Sistemas Computacionales

MATERIA:

Patrones De Diseño De Software

Nombre del trabajo

1.3 Categorías o clasificación de los patrones de diseño

UNIDAD A EVALUAR:

Unidad 1

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

González Guzmán María José #19211650

NOMBRE DEL MAESTRO (A):

Jose De Jesus Parra Galaviz

Fecha de entrega: 10 de septiembre del 2023

Categorías de los patrones de diseño

Patrones de diseño creacional: Los patrones de diseño creacionales abstraen el proceso de instanciación. Ayudan a que un sistema sea independiente de cómo se crean, componen y representan sus objetos. Un patrón creacional de clase utiliza la herencia para variar la clase que se instancia, mientras que un patrón creacional de objeto delegará la instanciación a otro objeto.

- Abstract Factory (Objeto): Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.
- Builder (Objeto): Separa la construcción de un objeto complejo de su representación para que el mismo proceso de construcción pueda crear diferentes representaciones.
- Factory Method (Clase): Define una interfaz para crear un objeto, pero permite a las subclases decidir qué clase instanciar. El Factory Method permite que una clase retrase la instanciación a las subclases.
- Prototype (Objeto): Especifica los tipos de objetos a crear utilizando una instancia prototípica y crea nuevos objetos copiando este prototipo.
- Singleton (Objeto): Asegura que una clase solo tenga una instancia y proporciona un punto de acceso global a ella.

Patrones Estructurales: se preocupan por cómo se componen las clases y objetos para formar estructuras más grandes. Los patrones estructurales de clases utilizan la herencia para componer interfaces o implementaciones. Como ejemplo simple, considera cómo la herencia múltiple mezcla dos o más clases en una. El resultado es una clase que combina las propiedades de sus clases padres. Este patrón es especialmente útil para hacer que las bibliotecas de clases desarrolladas de manera independiente trabajen juntas. En lugar de componer interfaces o implementaciones, los patrones estructurales de objetos describen formas de componer objetos para lograr nuevas funcionalidades. La flexibilidad añadida de la composición de objetos proviene de la capacidad de cambiar la composición en tiempo de ejecución, lo cual es imposible con la composición estática de clases.

- Adapter (Clase/Objeto): Convierte la interfaz de una clase en otra interfaz que los clientes esperan. El Adaptador permite que las clases trabajen juntas incluso si sus interfaces no son compatibles.

- Bridge (Objeto): Desacopla una abstracción de su implementación para que ambas puedan variar de forma independiente.
- Composite (Objeto): Compone objetos en estructuras de árbol para representar jerarquías parte-todo. El Composite permite que los clientes traten objetos individuales y composiciones de objetos de manera uniforme.
- Decorator (Objeto): Adjunta responsabilidades adicionales a un objeto de forma dinámica. Los decoradores proporcionan una alternativa flexible a la herencia para extender la funcionalidad.
- Facade (Objeto): Proporciona una interfaz unificada para un conjunto de interfaces en un subsistema. La Fachada define una interfaz de nivel superior que hace que el subsistema sea más fácil de usar.
- Flyweight (Objeto): Utiliza el uso compartido para admitir grandes cantidades de objetos de granularidad fina de manera eficiente.
- Proxy (Objeto): Proporciona un sustituto o marcador de posición para otro objeto para controlar el acceso a él.

Patrones de Comportamiento: se ocupan de los algoritmos y la asignación de responsabilidades entre objetos. Los patrones de comportamiento no solo describen patrones de objetos o clases, sino también los patrones de comunicación entre ellos. Estos patrones caracterizan flujos de control complejos que son difíciles de seguir en tiempo de ejecución. Desplazan tu enfoque lejos del flujo de control para que puedas concentrarte únicamente en la forma en que los objetos están interconectados.

- Chain of Responsibility (Objeto): Evita el acoplamiento entre el emisor de una solicitud y su receptor al darle a más de un objeto la oportunidad de manejar la solicitud. Encadena los objetos receptores y pasa la solicitud a lo largo de la cadena hasta que un objeto la maneja.
- Command (Objeto): Encapsula una solicitud como un objeto, lo que permite parametrizar clientes con diferentes solicitudes, encolar o registrar solicitudes y admitir operaciones deshacer.
- Interpreter (Clase): Dado un lenguaje, define una representación para su gramática junto con un intérprete que utiliza la representación para interpretar oraciones en el lenguaje.
- Iterator (Objeto): Proporciona una forma de acceder a los elementos de un objeto agregado de manera secuencial sin exponer su representación subyacente.

- Mediator (Objeto): Define un objeto que encapsula cómo un conjunto de objetos interactúa. El Mediador promueve el acoplamiento débil al evitar que los objetos se refieran entre sí explícitamente y permite variar su interacción de manera independiente.
- Memento (Objeto): Sin violar la encapsulación, captura y externaliza el estado interno de un objeto para que el objeto pueda restaurarse a este estado más tarde.
- Observer (Objeto): Define una dependencia uno a muchos entre objetos para que cuando un objeto cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente.
- State (Objeto): Permite que un objeto cambie su comportamiento cuando su estado interno cambia. El objeto parecerá cambiar de clase.
- Strategy (Objeto): Define una familia de algoritmos, encapsula cada uno y los hace intercambiables. La Estrategia permite que el algoritmo varíe de manera independiente de los clientes que lo utilizan.
- Template Method (Clase): Define el esqueleto de un algoritmo en una operación, aplazando algunas etapas a las subclases. El Template Method permite que las subclases redefinan ciertas etapas del algoritmo sin cambiar su estructura.
- Visitor (Objeto): Representa una operación que se realizará en los elementos de una estructura de objetos. El Visitor permite definir una nueva operación sin cambiar las clases de los elementos en los que opera.

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method (107)	Adapter (class) (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (object) (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Flyweight (195) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Observer (293) State (305) Strategy (315) Visitor (331)

Table 1.1: Design pattern space

(Gamma et al., 1994)

Bibliografía

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.