UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR (UTB)

Patrones de diseño

Grupo: E

Integrantes:

Guzman Perez Esneider Enrique – T00061918

Martinez Palmieri Kevin Esteban - T00061428

Quintana Fajardo Diego Andres - T 00061882

Curso:

Arquitectura de software – 2303

Número de documento:	Revisión:	Fecha de revisión
•	•	•
•	•	•
•	•	•
•	•	•
•	•	•
•	•	•

Contenido

Tabla de ilustraciones

1 Patrones de diseño creacionales

Son aquellos que proporcionan varios mecanismos de creación de objetos que incrementan la flexibilidad y reutilización de código existente. Estos tipos de patrones se dividen en varios los cuales son: Factory Method, Abstract Factory, Builder, Prototype y, por último, pero no menos importante se encuentra el Singleton.

En esta sección del documento los autores no tocaran los diferentes tipos de patrones de diseños creacionales con la excepción del patrón Builder. El cual será estudiado a profundidad en este documento.

1.1. Patrón de diseño Builder

El patrón de diseño Builder se es basado en la construcción de objetos complejos paso a paso. Este patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción. Este tipo de patrones ayuda a mejorar la seguridad de construcción como la legibilidad del código del programa. El objetivo general de este tipo de patrones es de crear varias representaciones utilizando un solo constructor, de esta forma se elimina una posible duplicidad en códigos que hacen cosas iguales o parecidas.

Como lo indican los autores del libro [1] “Separar la construcción de un objeto complejo de su representación para que este mismo proceso de construcción pueda crear diferentes representaciones”

Tomando como base lo anterior podemos observar en la imagen 1.0 lo que podría corresponder a este tipo de patrón de diseño, esto se debe a que como es una parte secuencial basándose en pasos que permiten construir diferentes representaciones basándose en una principal

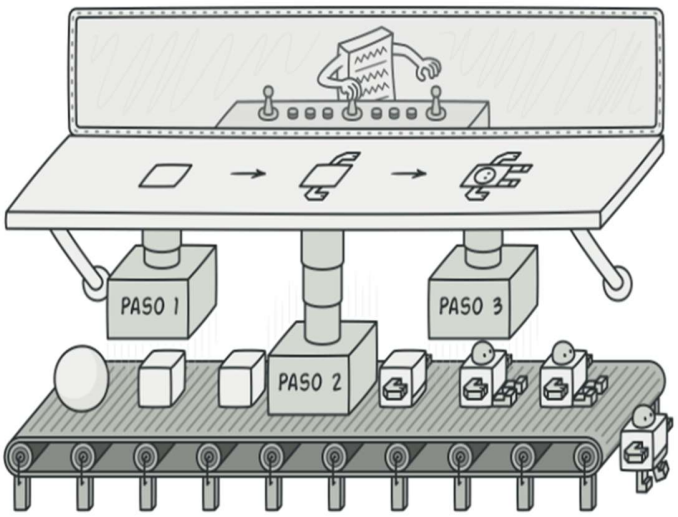

Ilustración 1 Representación gráfica del patrón de diseño builder.

1.1.1. Actores en el patrón de diseño builder.

En este tipo de patrón de diseño pueden identificar cuatro tipos diferentes de actores, los cuales cumplen diferentes acciones que son indispensable para que el código funcione de forma idónea, estos actores son los siguientes:

- ✓ **Director:** Este actor es el "constructor" del objeto complejo utilizando la interfaz o la información que maneja el actor constructor. Este actor es capaz de reconocer los requisitos de secuencia del constructor. Con el constructor, se desvincula la construcción del objeto del cliente.
  Este actor es también el que supervisa el proceso decisivo en el patrón.
- ✓ **Builder (Constructor):** Es el actor que provee la interfaz o información para la creación de los componentes de un objeto (o producto) complejo.
- ✓ **Specific Builder:** Es el encargado de crear las partes del objeto complejo. También define, gestiona la representación del objeto y es capaz de mantener la interfaz de salida del objeto construido.
- ✓ **Producto:** Es el resultado final de la "Actividad" del patrón de diseño en otras palabras es el objeto que se construyó utilizando el constructor base.

1.1.2. Diagrama UML

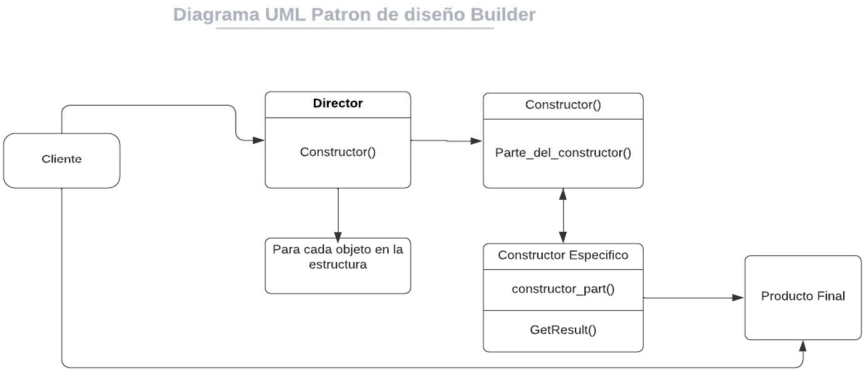

Ilustración 2 Diagrama UML patrón de diseño builder.

1.1.3. Ventajas y desventajas.

Todos los patrones de diseño tienen sus ventajas y desventajas, en esta sección estableceremos las ventajas y desventajas principales en el patrón de diseño builder. Para

poder establecer estas ventajas y desventajas serán utilizados subtítulos, estas ventajas y desventajas fueron extraídas de [2] .

1.1.3.1. Ventajas.

Las nuevas representaciones como tal pueden integrarse fácilmente utilizando las clases de constructores concretos, lo que nos permite evitar sobrecargar un constructor con información que no se necesitara.

Los cambios que se desean realizar se pueden hacer sin necesitar consultar a los clientes.

1.1.3.2. Inconvenientes.

El patrón constructor consta de fuerte vinculo entre el producto, el constructor especifico y las clases del proceso de diseño, así que puede ser difícil hacer cambios en el proceso básico. La construcción de los objetos requiere conocer su uso y su entorno en concretos.

1.1.4. Ejemplo práctico.

Para visualizar de forma más clara este patrón de diseño, los autores de este documento tomaron la decisión de mostrar un caso donde se pueda apreciar este. Por tanto, el caso de estudio donde se puede apreciar es el de un restaurante donde el cliente hace un pedido. Una vez que han recibido el pedido, los cocineros actúan para elaborarlo. Todo el proceso hasta la entrega como tal se hace "entre bambalinas"; el cliente no ve lo que ocurre en la cocina y solo recibe el resultado. En la ilustración número tres se puede observar el proceso como patrón de diseño.

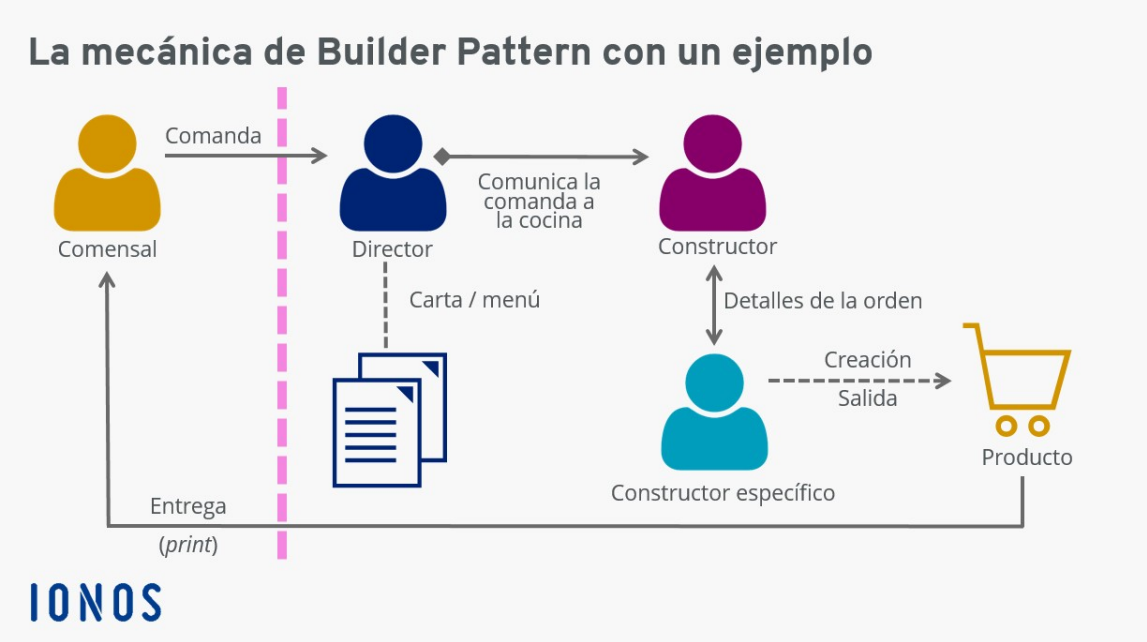

Ilustración 3 Ejemplo de patrón de builder.

2. Patrones de diseño estructurales

2.1. Patrón de diseño Flyweight

Este patrón nos permite que podamos usar más objetos dependiendo la capacidad disponible de la RAM que comparten las partes comunes del estado entre los diferentes objetos. Esta forma evita que en lugar de almacenar toda la información en cada objeto la podamos compartir. [3], En la imagen número cuatro se puede observar de forma en cómo trabaja este tipo de patrón de diseño

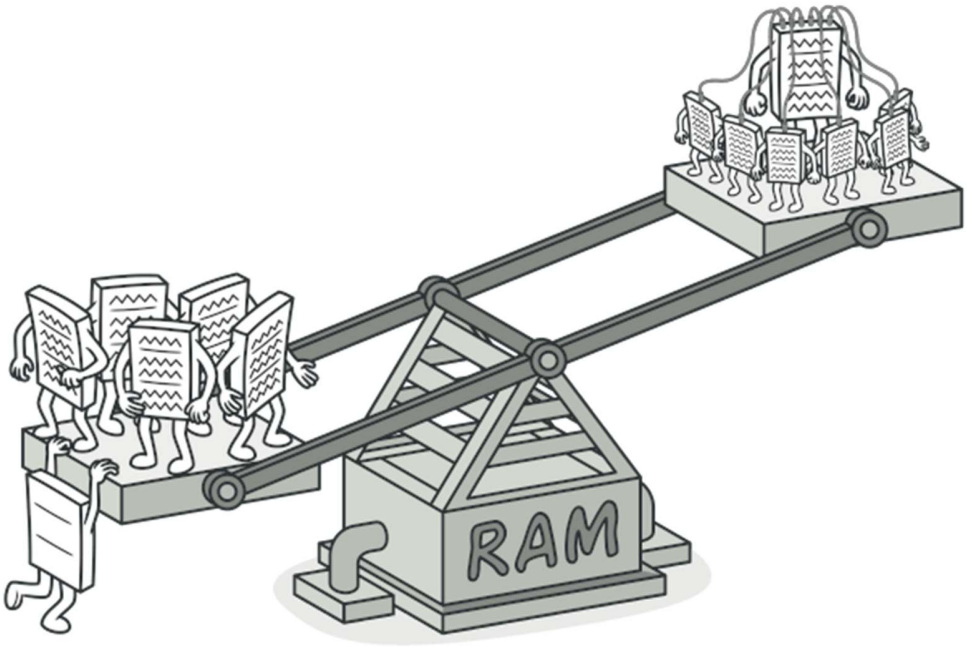

Ilustración 4 Ilustración de la forma en cómo funciona el patrón de diseño Flyweight

2.1.1. Diagrama UML

En la ilustración cinco se puede observar el diagrama de clase utilizado en este tipo de patrones.

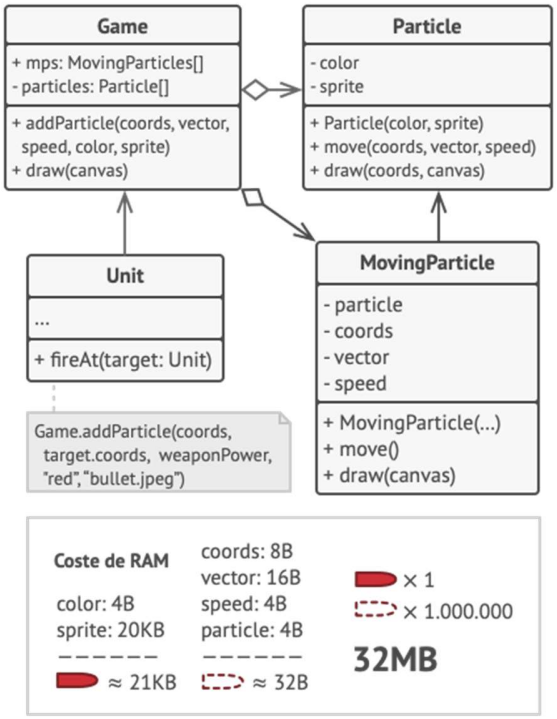

Ilustración 5 Diagrama UML del patrón estructural Flyweigth

2.1.2. Ventajas y desventajas

Como ya se dijo anteriormente los patrones de diseño tiene sus pro y contras y este modelo Flyweight no es la excepción, a continuación, veremos sus ventajas y desventajas

2.1.2.1. Ventajas

El modelo Flyweight como ya sabemos sirve más que todo para optimizar, gracias a este se puede ahorrar la RAM. Esto siempre y cuando gamos objetos similares [3]

2.1.2.2. Desventajas

Una de las desventajas es que "El código se complica mucho" ¿Por qué? Porque los miembros nuevos se preguntarán "¿por qué estados de una entidad se separó de esa forma?" [3]

3. Patrones de comportamiento.

Son diseño que manejan la comunicación de diferentes partes en el código de una aplicación, comúnmente se encargan de la interacción que tiene cada objeto con otros. Al hacerlo, estos patrones mejoran significativamente la escalabilidad y flexibilidad de la comunicación de estos.

De estos tipos de patrones podemos mencionar algunos como: Command, Iterator, Mediator… Etc. Específicamente, en este documento hablaremos sobre el patrón de diseño Mediator.

3.1.1. Patrón de diseño Mediator.

Este patrón de diseño se especializa restringir la comunicación entre los diversos objetos de la aplicación en desarrollo, obligando a estos, comunicarse a través de una sola clase que se encarga de ser el mediador de la comunicación entre estás, como se ilustra en la figura 4. Esto evita tener múltiples llamadas de clases al tener un orden y control más consistente al momento de la ejecución de la aplicación, además, garantiza una mayor escalabilidad de esta misma, ya que se vuelve mucho más sencilla las conexiones de futuros componentes.

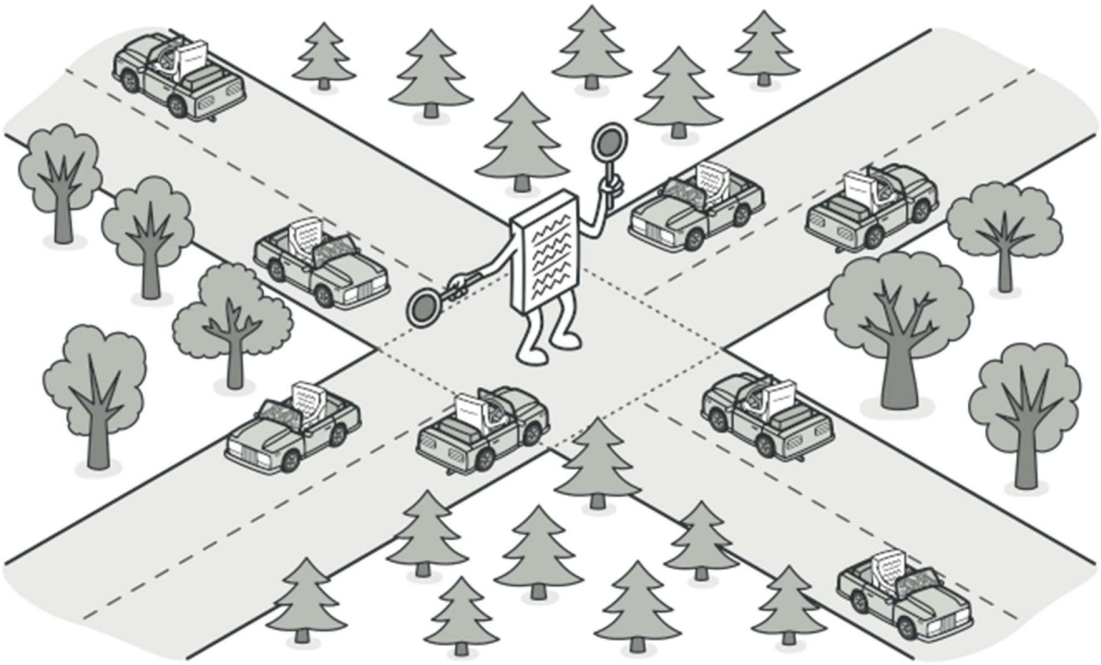

Ilustración 6: Representación gráfica del patrón de diseño mediador.

3.1.2. Actores en el patrón de de diseño Mediator.

En este tipo de patrón de diseño pueden identificar tres tipos diferentes de actores, los cuales se encargan de gestionar y garantizar el buen rendimiento de forma idónea del patrón:

- ✓ **Componentes:** Este actor son el conjunto de clases que tienen la lógica del negocio, y cada uno de estos, tienen como referencia a las interfaces mediadoras, pero realmente, no tiene una clase mediadora definida para usar, lo cual se podría usar estos componentes en cualquier otro proyecto de forma fácil y rápida.
- ✓ **Interfaz mediadora:** Es el actor encargado de la comunicación entre los demás componentes, puede aceptar cualquier contexto como parámetro en la interfaz mediadora, de tal forma que no pueda haber otro método de interacción con las demás componentes del software.

- ✓ **Mediadores concretos:** Se encarga de la encapsulación de los componentes respecto a otros de estos mismos, y mantienen la relación indirecta de todos ellos entre sí, y ocasionalmente, pueden gestionar incluso, el ciclo de vida de estos.

3.1.3. Diagrama de clase mediador.

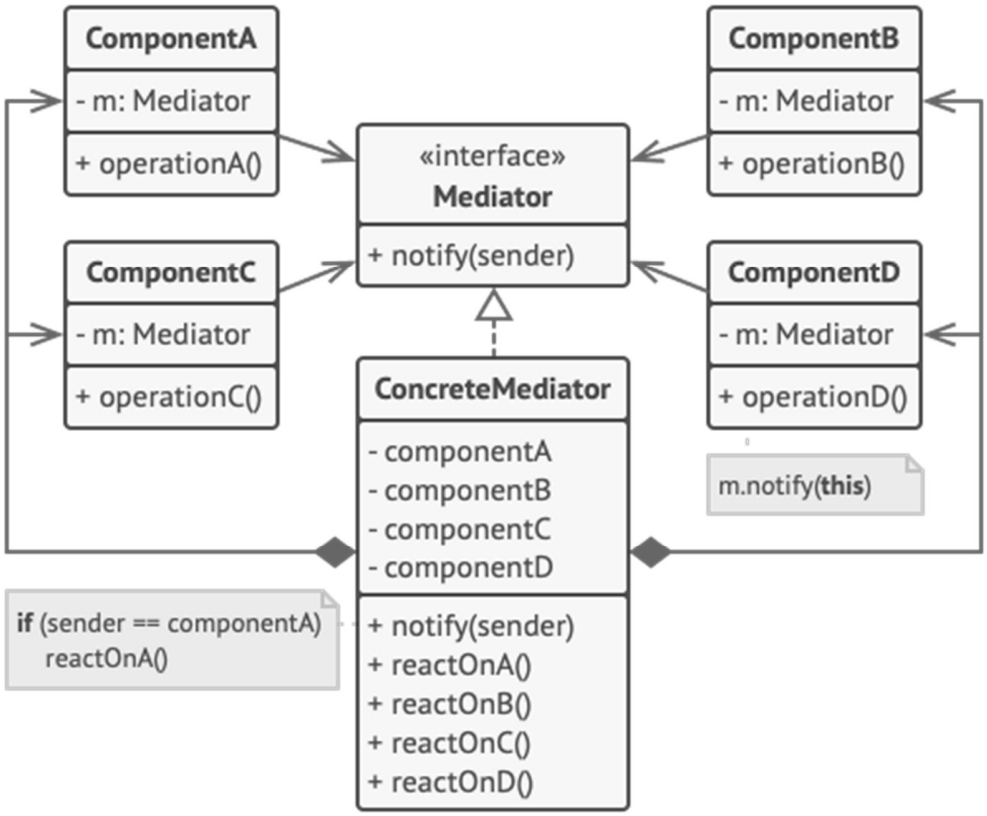

Ilustración 7: Diagrama de clase mediator.

3.1.4. Ventajas y desventajas.

Como todo patrón, tiene sus ventajas y desventajas al usarlo, por ende, en esta sección describiremos esto mismo:

3.1.4.1. Ventajas.

Permite optimizar la creación de componente, centralizar la interacción de los diversos componentes y permite la implementación de nuevos y reutilización de estos mismos, para proyectos diferentes.

3.1.4.2. Desventajas.

Al momento de desarrollar la clase mediadora en el proyecto, se puede volver muy complicada de entender, y llegara un punto donde puede que se complique darle mantenimiento.

4. Anexos

En esta sección se procederá a mostrar o indicar el link necesario para mostrar los códigos de los patrones de diseño que en este documento fueron presentados.

Link de GitHub

- ✓ https://github.com/Esneider23/Patrones.git

Referencias

[1] E. Gamma, R. Helm, J. Ralph y J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, New york: Addison-Wesley Professional, 1994.

[2] IONOS, «IONOS DIGITAL GUIDE,» IONOS, 21 FEBRERO 2022. [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/patron-de-diseno-builder/. [Último acceso: 7 SEPTIEMBRE 2022].