Aprendiz: Rafael Dario Escalante Sandoval

Equipo de proyecto:

- Saray Acosta
- Danny Alexander Minota Soto
- Cristina Mosquera Rodriguez

Instructor: Andrés Rubiano Cucarian

desarrollar la arquitectura de software de acuerdo con el patrón de diseño seleccionado GA4-220501095-AA2-EV05.

ANALISIS Y DESARROLLO DE SOFTWARE. (2977466)

Contents

Introduccion

Los **patrones de diseño** son soluciones habituales a problemas que ocurren con frecuencia en el diseño de software. Son **como planos prefabricados** que se pueden personalizar para resolver un problema de diseño recurrente en tu código.

No se puede elegir un patrón y copiarlo en el programa como si se tratara de funciones o bibliotecas ya preparadas. El patrón no es una porción específica de código, sino un **concepto general para resolver un problema particular.** Puedes seguir los detalles del patrón e implementar una solución que encaje con las realidades de tu propio programa.

A menudo los patrones se confunden con algoritmos porque ambos conceptos describen soluciones típicas a problemas conocidos. Mientras que un algoritmo siempre define un grupo claro de acciones para lograr un objetivo, **un patrón es una descripción de más alto nivel de una solución.** El código del mismo patrón aplicado a dos programas distintos puede ser diferente.

Una analogía de un algoritmo sería una receta de cocina: ambos cuentan con pasos claros para alcanzar una meta. Por su parte, un patrón es más similar a un plano, ya que puedes observar cómo son su resultado y sus funciones, pero el orden exacto de la implementación depende de ti.

Patrones de diseño

Patrones creacionales

Los patrones de creación proporcionan diversos mecanismos de creación de objetos, que aumentan la **flexibilidad y la reutilización del código existente de una manera adecuada a la situación**. Esto le da al programa más flexibilidad para decidir qué objetos deben crearse para un caso de uso dado.

Estos son los **patrones creacionales**:

Abstract Factory

En este patrón, una interfaz crea conjuntos o familias de objetos relacionados sin especificar el nombre de la clase.

Builder Patterns

Permite producir diferentes tipos y representaciones de un objeto utilizando el mismo código de construcción. Se utiliza para la creación etapa por etapa de un objeto complejo combinando objetos simples. La creación final de objetos depende de las etapas del proceso creativo, pero es independiente de otros objetos.

Factory Method

Proporciona una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de objetos que se crearán. Proporciona instanciación de objetos implícita a través de interfaces comunes

Prototype

Permite copiar objetos existentes sin hacer que su código dependa de sus clases. Se utiliza para restringir las operaciones de memoria / base de datos manteniendo la modificación al mínimo utilizando copias de objetos.

Singleton

Este patrón de diseño restringe la creación de instancias de una clase a un único objeto.

Patrones estructurales

Facilitan soluciones y estándares eficientes con respecto a las composiciones de clase y las estructuras de objetos. El concepto de herencia se utiliza para componer interfaces y definir formas de componer objetos para obtener nuevas funcionalidades.

Adapter

Se utiliza para vincular dos interfaces que no son compatibles y utilizan sus funcionalidades. El adaptador permite que las clases trabajen juntas de otra manera que no podrían al ser interfaces incompatibles.

Bridge

En este patrón hay una alteración estructural en las clases principales y de implementador de interfaz sin tener ningún efecto entre ellas. Estas dos clases pueden desarrollarse de manera independiente y solo se conectan utilizando una interfaz como puente.

Composite

Se usa para agrupar objetos como un solo objeto. Permite componer objetos en estructuras de árbol y luego trabajar con estas estructuras como si fueran objetos individuales.

Decorator

Este patrón restringe la alteración de la estructura del objeto mientras se le agrega una nueva funcionalidad. La clase inicial permanece inalterada mientras que una clase *decorator* proporciona capacidades adicionales.

Facade

Proporciona una interfaz simplificada para una biblioteca, un marco o cualquier otro conjunto complejo de clases.

Flyweight

El patrón Flyweight se usa para reducir el uso de memoria y mejorar el rendimiento al reducir la creación de objetos. El patrón busca objetos similares que ya existen para reutilizarlo en lugar de crear otros nuevos que sean similares.

Proxy

Se utiliza para crear objetos que pueden representar funciones de otras clases u objetos y la interfaz se utiliza para acceder a estas funcionalidades

Patrones de comportamiento

El patrón de comportamiento se ocupa de la **comunicación entre objetos de clase.** Se utilizan para detectar la presencia de patrones de comunicación ya presentes y pueden manipular estos patrones.

Estos patrones de diseño están específicamente relacionados con la comunicación entre objetos.

Chain of responsibility

El patrón de diseño Chain of Responsibility es un patrón de comportamiento que evita acoplar el emisor de una petición a su receptor dando a más de un objeto la posibilidad de responder a una petición.

Command

Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación permite parametrizar métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y respaldar operaciones que no se pueden deshacer.

Interpreter

Se utiliza para evaluar el lenguaje o la expresión al crear una interfaz que indique el contexto para la interpretación.

Iterator

Su utilidad es proporcionar acceso secuencial a un número de elementos presentes dentro de un objeto de colección sin realizar ningún intercambio de información relevante.

Mediator

Este patrón proporciona una comunicación fácil a través de su clase que permite la comunicación para varias clases.

Memento

El patrón *Memento* permite recorrer elementos de una colección sin exponer su representación subyacente.

Observer

Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que está siendo observado.

State

En el patrón *state*, el comportamiento de una clase varía con su estado y, por lo tanto, está representado por el objeto de contexto.

Strategy

Permite definir una familia de algoritmos, poner cada uno de ellos en una clase separada y hacer que sus objetos sean intercambiables.

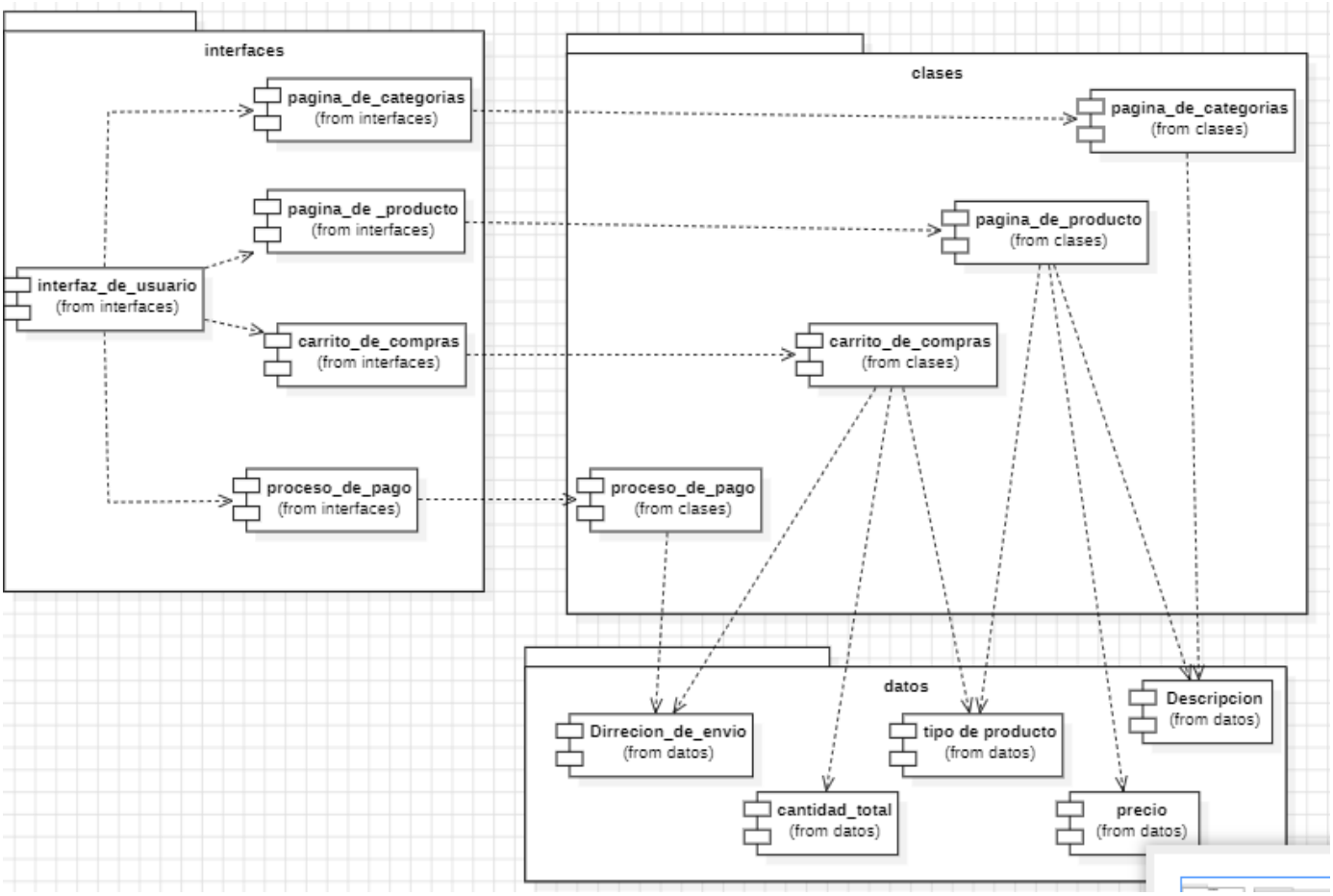
Template method

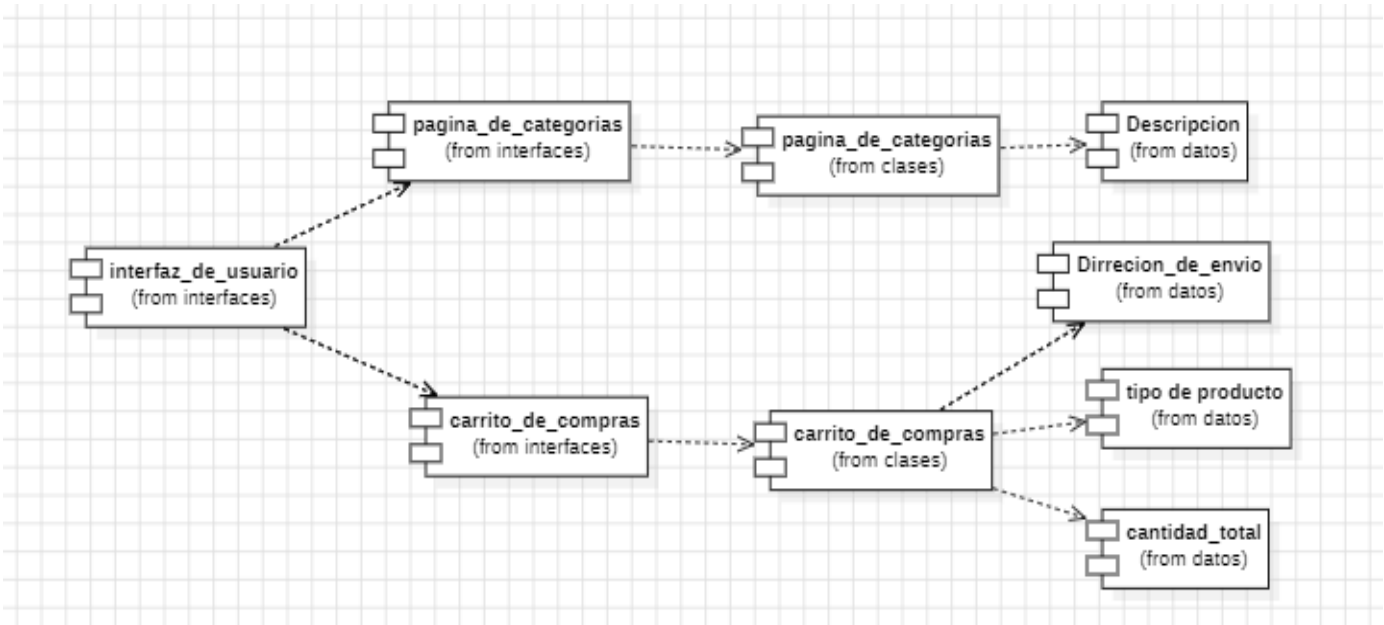
Se usa con componentes que tienen similitud donde se puede implementar una plantilla del código para probar ambos componentes. El código se puede cambiar con pequeñas modificaciones.

Visitor

El propósito de un patrón Visitor es definir una nueva operación sin introducir las modificaciones a una estructura de objeto existente.

Diagrama general de componente





Conclusión

Un **diagrama de componentes** en UML (Lenguaje Unificado de Modelado) es una herramienta visual que nos permite representar la estructura física de un sistema, mostrando cómo se descompone en partes más pequeñas y cómo interactúan entre sí.

Fuentes

https://profile.es/blog/patrones-de-diseno-de-software/

https://youtu.be/56-W0pWmEM8?si=XoWmqExqq7vY-9jN

https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

https://zajuna.sena.edu.co/Repositorio/Titulada/institution/SENA/Tecnologia/228118/Contenido/OVA/CF18/index.html#/curso/tema5