Tecnológico Nacional de México
Instituto Tecnológico de Tijuana
Subdirección Académica
Departamento de Sistemas y Computación
Periodo: Enero-Junio 2020

Abstract Factory

Integrantes:

- Aguiar Solis Pablo 16211958
- Navarro González Esmeralda 16212052

Fecha de entrega
Tijuana,B.C. 10 de Febrero del 2020.

Materia: Patrones de diseño.
Serie: ADF-1701 SC8B.
Docente: Martha Elena Pulido.

Introducción

Los patrones de diseño tienen su origen en la Arquitectura, cuando en 1979 el Arquitecto Christopher Alexander publicó el libro Timeless Way of Building, en el cual hablaba de una serie de patrones para la construcción de edificios.

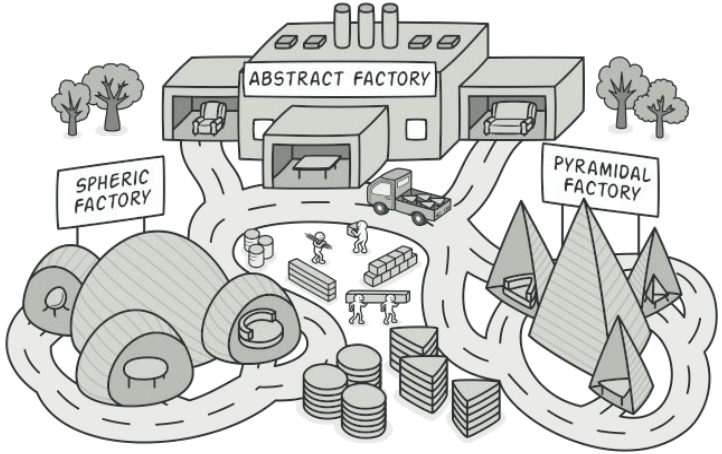
Él utilizaba las siguientes palabras: "Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez."

Patrones creacionales

Los patrones creacionales sirven para controlar la forma en que se crean los objetos, de entrada puede parecer un poco extraño, ya que se tiene la costumbre de crear libremente los objetos, sin embargo, existen situaciones donde por conveniencia es necesario establecer un mecanismo que permita crear instancias de una forma controlada.

Abstract factory

Provee una interfaz para crear familias de objetos relacionados o dependientes entre ellos sin especificar una clase en concreto. Los patrones de Abstract Factory funcionan en torno a una súper fábrica que crea otras fábricas.



Características

1. Aísla clases concretas. El patrón Abstract Factory ayuda a controlar las clases de objetos que crea una aplicación.

2. Facilita el intercambio de familias de productos. La clase de una fábrica concreta aparece solo una vez en una aplicación, es decir, donde se instancia. Esto facilita cambiar la fábrica concreta que utiliza una aplicación.

3. Promueve la consistencia entre los productos. Cuando los objetos de productos en una familia están diseñados para trabajar juntos, es importante que una aplicación use objetos de una sola familia a la vez.

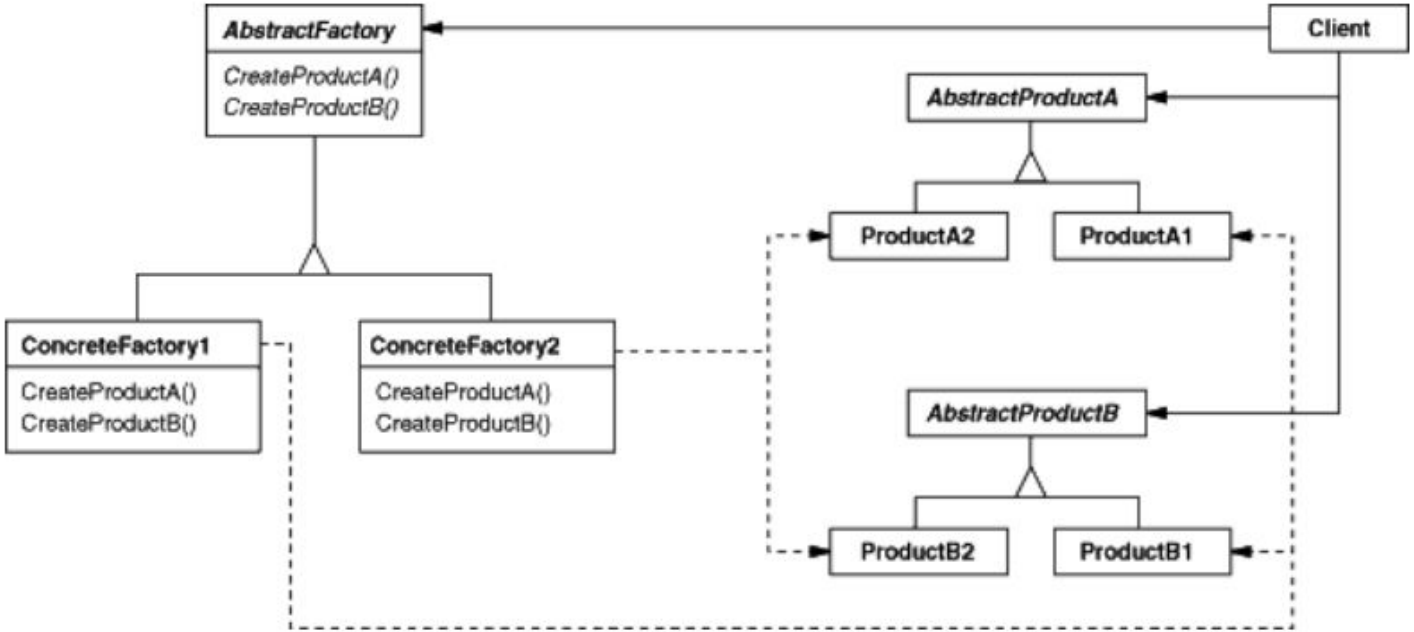
Metodología

- Decidir si la instancia de los objetos y la creación de servicios son la fuente de nuestros problemas.
- Definir los productos abstractos.
- Definir los productos concretos.
- Definir las fábricas concretas o familia de productos.
- Definir la fábrica abstracta.
- Implementar las llamadas o instancias de objetos desde el cliente.
- Verificar la salida.

Ventajas y desventajas

Ventajas	Desventajas
Principio de responsabilidad única. Puede extraer el código de creación del producto en un solo lugar, haciendo que el código sea más fácil de soportar.	El código puede volverse más complicado de lo que debería ser, ya que se introducen muchas nuevas interfaces y clases junto con el patrón.
Evita el acoplamiento estrecho entre productos concretos y el código del cliente.	Difícil de agregar nuevos tipos de productos: No es fácil ampliar las fábricas abstractas para producir nuevos tipos de productos. Esto se debe a que la interfaz Abstract Factory corrige el conjunto de productos que se pueden crear. La compatibilidad con nuevos tipos de productos requiere ampliar la interfaz de la fábrica, lo que implica cambiar la clase Abstract Factory y todas sus subclases.
Los productos que obtienen de una fábrica son compatibles entre sí.	
Principio abierto / cerrado. Puede introducir nuevas variantes de productos sin romper el código de cliente existente.	

Componentes y estructura

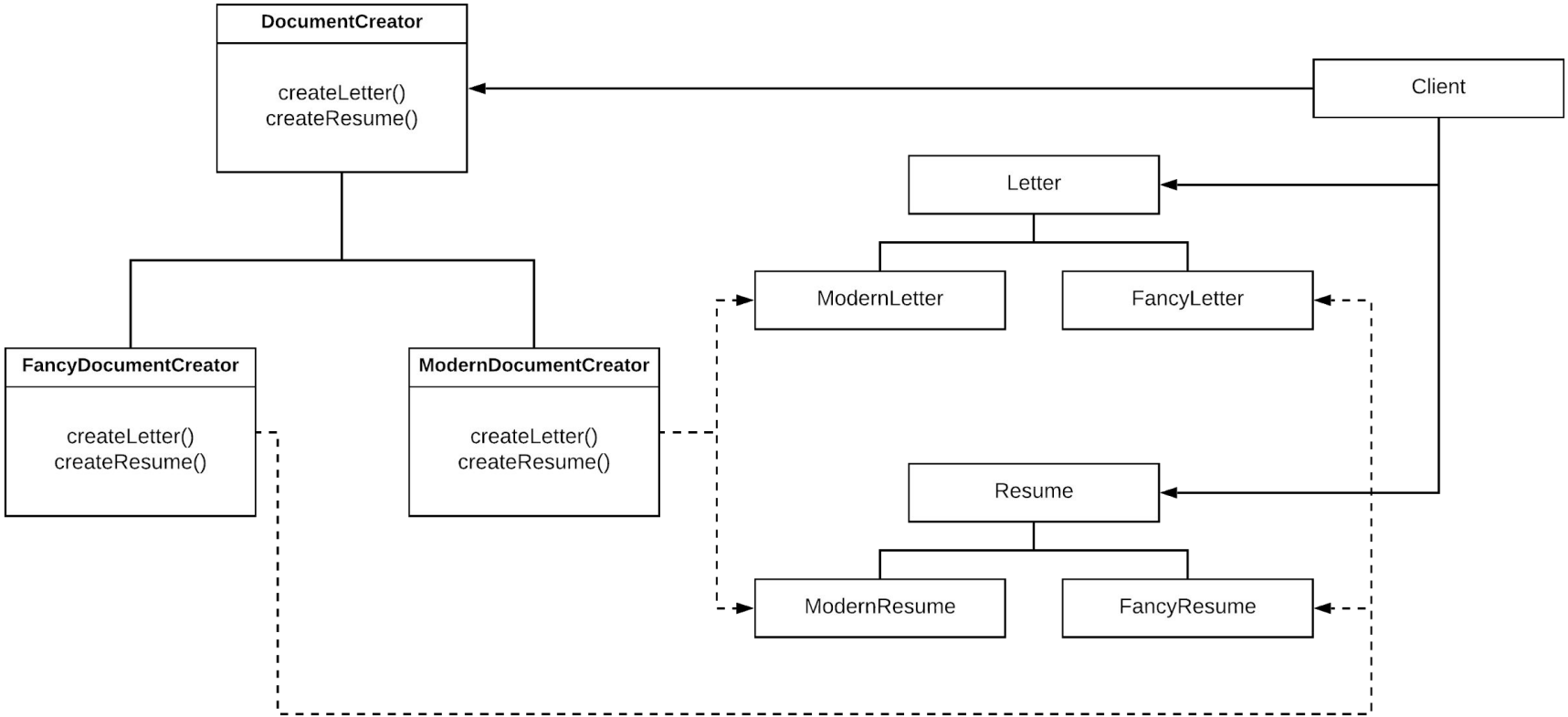


Aplicaciones

- Cuando el sistema necesita ser independiente de cómo se crea, compone y representa su objeto.
- Cuando la familia de objetos relacionados tiene que usarse en conjunto, entonces esta restricción debe hacerse cumplir.
- Cuando desee proporcionar una biblioteca de objetos que no muestre implementaciones y solo revele interfaces.
- Cuando el sistema necesita ser configurado con uno de una familia múltiple de objetos.

Ejemplo

Práctica


DocumentCreator
createLetter()
createResume()
Client
Letter
ModernLetter
FancyLetter
FancyDocumentCreator
createLetter()
createResume()
ModernDocumentCreator
createLetter()
createResume()
Resume
ModernResume
FancyResume

Conclusión

Con el desarrollo de este documento e investigación, se concluye que los patrones de diseño resultan ser muy importantes, debido a que ayudan a estandarizar el código, haciendo que el diseño sea más comprensible para otros programadores. Son muy buenas herramientas, y como programadores, siempre deberíamos usar las mejores herramientas a nuestro alcance.

Referencias

[1] Oscar Javier Blancarte (2016). Introducción a los Patrones de Diseño. Ciudad de México, México.

[2] John Vlissides, Ralph Johnson, Richard Helm, Erich Gamma (1994). Design Patterns: Elements of Reusable Object-Oriented Software

[3] Ecured.”Abstract factory”. Consultado en: https://www.ecured.cu/Abstract_Factory

[4] Javatpoint. ”Abstract Factory Pattern”. Consultado en: https://www.javatpoint.com/abstract-factory-pattern