

Tarea #1

Autores:

Josselyn Mora Santamaria

[josselyn.mora.santamaria@est.una.ac.cr](mailto:josselyn.mora.santamaria@est.una.ac.cr)

Joshua Granados Loría

[joshua.granados.loria@est.una.ac.cr](mailto:joshua.granados.loria@est.una.ac.cr)

Kevin Mora Valverde

[kevin.mora.valverde@est.una.ac.cr](mailto:kevin.mora.valverde@est.una.ac.cr)

Universidad Nacional de Costa Rica.

Curso: Programación III

II Ciclo-NCR 51172-Grupo 89

Profesor: Máster Rubén Mora Vargas

04 de octubre del 2021

Tabla de contenido

[Introducción 3](#_Toc84063250)

[Contenido 4](#_Toc84063251)

[Abtract Factory 4](#_Toc84063252)

[Flyweight 4](#_Toc84063253)

[Memento 6](#_Toc84063254)

[Conclusión 7](#_Toc84063255)

[Recomendaciones 8](#_Toc84063256)

[Bibliografías 9](#_Toc84063257)

[Anexos 10](#_Toc84063258)

# Introducción

# Contenido

## Abtract Factory

El abstract factory o fábrica abstracta en español, es parte de los patrones de diseño propuestos por el Gang of Four (Gamma, Helm, Johnson y Vlissides) los cuales se utilizan como una guía que los programadores utilizan acorde a las funciones o exigencias de la aplicación que deseen desarrollar (Blanco & Universidad de Cantabria, s.f.).

Dicho patrón de diseño crea una sola interfaz para crear una familia completa de productos, el usuario que la utiliza solo ve la interfaz, no ocupa conocer las clases que la componen. Además, es conocida con el nombre de kit.

Abstract factory es un patrón de diseño con calificación creacional con un ámbito de objetos, es decir actúa como una fábrica que crea objetos relacionados entre sí o que dependan entre ellos. También realiza una encapsulación de reglas de instanciación, esta permite que en la clase abstract factory se puedan ocultar el proceso para la instanciación de un grupo de objetos (Polanco, F. G., s.f.).

La motivación al utilizar dicho patrón es poder crear aplicaciones independientes de la interfaz con la cual el usuario interactúa.

Además, es ideal para cuando una aplicación requiere manejar muchas familias de productos.

Según la Universidad Carlos III de Madrid (2007) para llevar a cabo la creación de una fábrica abstracta se debe establecer lo siguiente:

**AbstractFactory**: Es la interfaz encargada de crear los objetos abstractos.

**ConcreteFactory:** Implementa las operaciones para crear objetos concretos.

**AbstractProduct:** Define la interfaz de un objeto producto.

**ConcreteProduct:** Implementa la interfaz AbstractProduct.

**Client:** Utiliza las interfaces de AbstractFactory y AbstractProduct.

La siguiente información fue obtenida por medio del estudio de Blanco y Universidad de Cantabria (s.f.):

**Ventajas:**

* Facilita el cambio de familia de productos. Ya que la encargada de la familia de productos es la clase concrete y esta solo aparece una vez en el código.
* Permite que la aplicación pueda usar varios productos a la vez siempre y cuando sean de la misma familia.

**Desventajas:**

* Es complicado agregar nuevos productos, ya que afecta a la interfaz y por ende a las subclases.

Estructura del Abstract Factory:

En la figura 1 se puede observar un diagrama de cómo está compuesto el patrón de diseño AbstractFactory:

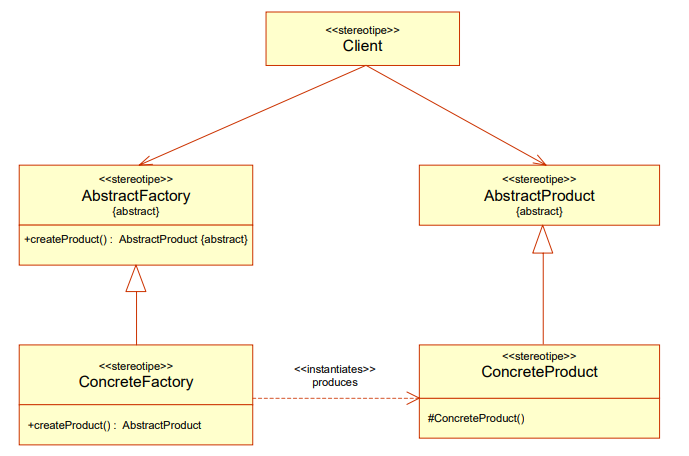
El cliente interactúa con las interfaces las cuales ayudaran al mismo en la utilización de la aplicación. La clase cliente accede a los métodos de ConcreteFactory por medio de la clase Abstract Factory. Y para utilizar los productos creados utiliza la interfaz AbstractProduct.

ConcreteFactory se encarga de implementar la interfaz AbstractFactory, posee las operaciones de crear y retornar objetos.

Y quien define los objetos creados es la ConcreteFactory.

Figura 1

Estructura del Abstract Factory



*Nota:* Estructura del patrón de diseño AbstractFactory. Pertenece al siguiente documento: *Abstract Factory* (Polanco, F. G., s.f.).

Diferencia entre Factory Method y Abstract Factory:

La diferencia entre estos dos métodos se basa en su objetivo, el abstract factory tiene como objetivo crear un conjunto de objetos relacionados entre sí, mientras que el Factory Method aplazar hacia una determinada clase o subclase el tipo de objeto que se va a instanciar.

## Flyweight

Este diseño de patrón sirve para reducir el uso de memoria, esto mediante la compartición de partes en común entre varios objetos, así se reduce una gran cantidad de código al no tener que mantener la misma información en todos los objetos, este proceso va mucho de la mano con lo que es la herencia ya que cumplen una función muy similar. El diseño de patrón Flyweight tiene la siguiente estructura:

1. Se tiene que entender que Flyweight es una manera de optimizar el código de la aplicación que se este creando, se suele implementar cuando el programa creado consume mucha RAM y no se logra disminuir ese consumo de otra manera.
2. Clase Flyweight: En esta clase se encuentra los estados del objeto original del cual los objetos hijos van a heredar, este estado que se almacena dentro del objeto Flyweight se le denomina como intrínseco y los que se les pasa a sus métodos se denomina extrínseco.
3. Clase Contexto: Dentro de esta clase se encuentra el estado extrínseco, este estado es único entre todos los objetos originales, en el momento en el que un contexto se logra unir con uno de los objetos Flyweight se logra un estado completo del objeto original.
4. Cuando se invoque un método de un objeto de tipo Flyweight se debe de pasar aquellas partes del estado que sean útiles mediante los parámetros del método, además los métodos que le dan un comportamiento al objeto Flyweight se pueden utilizar en la clase de contexto.
5. Clase cliente: En esta se va a realizar todo calculo o almacenamiento del estado extrínseco de los distintos objetos Flyweight. Desde una vista del propio cliente, un Flyweight es un tipo de objeto de tipo plantilla el cual puede ser configurado durante la ejecución del programa, esto pasa porque envía información contextual dentro de los parámetros de sus distintivos métodos.

1. Clase Fabrica Flyweight: Mediante esta clase se gestionan los grupos existentes de objetos Flyweight, por lo tanto, la clase cliente no se hace cargo de crear directamente los objetos Flyweight, si no que llama a la fabrica y se le envían los parámetros que se necesite del estado intrínseco. La fábrica hace una revisión de los objetos Flyweight que se crearon previamente y devuelve uno existente que pueda coincidir con los parámetros enviados y si no llega a encontrar ninguno entonces crea uno nuevo.

En la siguiente imagen se puede observar un diagrama UML con la estructura mencionada anteriormente.

Diagrama

Descripción generada automáticamente con confianza baja

Imagen#1. Diagrama UML de la estructura del patrón de diseño Flyweight

Ventajas de utilizar Flyweight:

* El ahorro del uso de la memoria RAM.

Desventajas:

* El código puede convertirse un poco complicado de entender.
* Si hay cambios de RAM por ciclos de CPU el calculo de una nueva parte de la información se realizará cada vez que se invoque un método Flyweight.

Para implementar este patrón de diseño en un código se debe de hacer de la siguiente manera:

1. Se tienen que dividir los campos de una clase para que se convierta en la clase Flyweight, esta clase se divide en dos partes:

* Estado intrínseco: Son aquellos campos que van a tener la información única de un objeto, esta va a estar duplicada a través de todos los objetos del mismo tipo.
* Estado extrínseco: Son los campos que almacenan toda información contextual que es única en cada objeto.

1. Los campos que incorporan el estado intrínseco se tienen que dejar en la clase sin embargo estos no deben de ser alterados, se tiene que asegurar que los valores iniciales se encuentren dentro del constructor.
2. Se tienen que revisar los métodos que utilizan el estado extrínseco para agregarles un nuevo parámetro y suplantar el campo.
3. Se crea la clase fabrica para encargarse de un grupo de objetos Flyweight, una vez se encuentre esta clase lista, los clientes podrán solicitar cualquier objeto Flyweight.
4. Dentro de la clase cliente se debe guardar o calcular los valores del estado extrínseco para así lograr la invocación de los métodos de los objetos Flyweight, si se desea se puede mover el estado extrínseco a la clase contexto. (Shvets, 2019)

## Memento

# Conclusión

# Recomendaciones

# Bibliografías

Shvets, A. (2019). *Flyweight*. Refactoring.Guru. <https://refactoring.guru/es/design-patterns/flyweight>

Kaisler, S. H. (2005). Software paradigms. ProQuest Ebook Central. <https://ebookcentral.proquest.com>

Polanco, F. G. (s.f.). *Abstract Factory (Fábrica Abstracta-GoF)* [Diapositivas]. <http://eii.ucv.cl/pers/guidi/cursos/estructuras/pdf/DesignPatternEs-001.pdf>

Universidad Carlos III de Madrid. (2007). *Patrones de diseño* [Diapositivas]. http://arantxa.ii.uam.es/~eguerra/docencia/0708/04%20Creacion.pdf

Blanco, C. & Universidad de Cantabria. (s.f.). *Patrones de diseño* [Diapositivas]. <https://ocw.unican.es/pluginfile.php/1403/course/section/1792/is1_t06_Patrones.pdf>

# Anexos

En el siguiente enlace de GitHub se puede encontrar un ejemplo programado dedicado a cada patrón de diseño investigado.

<https://github.com/joshua292000/Tarea-1.git>