Trabajo Práctico Nº1	Unidad 1
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo individual.
Metodología de Desarrollo: Det. docente	**Metodología de Corrección:** Via Classroom.
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

ALUMNA:

DOMINGUEZ, SANDRA

POO y Patrones TP1 – Patrones 001

Marco Teórico:

Responder el siguiente cuestionario en función de la bibliografía Obligatoria.

1. Describir los Tipos de Patrones.

Patrones arquitectónicos: son patrones usados en el diseño más general y el establecimiento de arquitecturas. Tienen mayor nivel de abstracción que un patrón de diseño y mayor impacto en la arquitectura en su conjunto.

Patrones de diseño: es un concepto y no una parte de código, que da solución a problemas frecuentes que ocurren en el diseño de software, éstos extienden y refinan el diseño de los patrones arquitectónicos, y se utilizan en el diseño más detallado. No se debe confundir a los patrones de diseño con los algoritmos, ya que son cosas diferentes. Proporcionan un esquema para refinar los elementos de un sistema de software o las relaciones entre ellos.

2. Que es un patrón de Software

Un patrón es una solución a un problema recurrente en el diseño de software. Se trata de utilizar la experiencia de otros programadores que ya pasaron por el mismo tipo de problemas generalizando una solución, con esto no se pretende utilizar en todos los casos patrones, sino saber que están y adaptarlos, si es posible, a las necesidades del problema en cuestión.

Un patrón de diseño proporciona un esquema para refinar los elementos de un sistema de software o las relaciones entre ellos. Describe una estructura comúnmente recurrente de elementos de diseño interconectados que resuelve un problema de diseño general dentro de un contexto particular.

Generalmente los patrones son descriptos de manera formal, en estas descripciones suelen detallarse el problema y la solución que ofrece, las estructura de sus clases y el modo en que se relacionan, y un ejemplo en algún lenguaje de programación popular. Pueden dar detalles utiles como aplicabilidad del patrón, pasos de implementación y relación con otros patrones.

3. Clasificar los Patrones de Software

Los patrones se clasificarse en tres grupos:

- **de creación** proveen mecanismos que solucionan o facilitan las tareas de creación o instanciación de objetos, que mejoran la flexibilidad y la reutilización de código existente.
- **estructurales** explican cómo ensamblar objetos y clases en estructuras más grandes a la vez que se mantiene la flexibilidad y eficiencia de la estructura.
- **de comportamiento** se encargan de una comunicación efectiva y la asignación de responsabilidades entre objetos.

4. clasificar los Patrones de Creación y realizar una Descripción

Constructor Separa la creación de un objeto complejo de su estructura, de tal forma que el mismo proceso de construcción nos puede servir para crear representaciones diferentes. Se utiliza un patrón de construcción para construir objetos. A veces, los objetos que creamos pueden ser complejos, estar formados por varios subobjetos o requerir un elaborado proceso de construcción. El ejercicio de crear tipos complejos se puede simplificar utilizando el patrón de construcción. Un objeto compuesto o agregado es lo que generalmente construye un constructor. El patrón Builder puede parecer similar al patrón de "fábrica abstracta", pero una diferencia es que el patrón Builder crea un objeto paso a paso, mientras que el patrón de fábrica abstracto devuelve el objeto de una vez.

Fábrica Expone un método de creación, delegando en las subclases la implementación de este método.

Fábrica abstracta Nos provee una interfaz que delega la creación de un conjunto de objetos relacionados sin necesidad de especificar en ningún momento cuáles son las implementaciones concretas.

Prototipo Permite la creación de objetos basados en «plantillas». Un nuevo objeto se crea a partir de la clonación de otro objeto.

Singleton limita a uno el número de instancias posibles de una clase en nuestro programa, y proporciona un acceso global al mismo. se utiliza para limitar la creación de una clase a un solo objeto. Esto es beneficioso cuando se necesita un objeto (y solo uno) para coordinar acciones en todo el sistema.

5. Dar un ejemplo de uso de cada uno.

Builder

Propósito: Separar la construcción de un objeto complejo de su representación para que el mismo proceso de construcción puede crear diferentes representaciones.

Ejemplo: Usamos el patrón Builder cuando queremos construir un objeto compuesto de otros objetos. O que la creación de las partes de un objeto sea independiente del objeto principal.

Singleton

Propósito: Asegurar que una clase tenga una única instancia y proporcionar un punto de acceso global a la misma. El cliente llama a la función de acceso cuando se requiere una referencia a la instancia única.

Ejemplo: Usamos el patrón Singleton cuando necesitamos Manejar conexiones con la base de datos, o la clase para hacer Login.

Factory o Factory Method

Propósito: Definir una interface para crear un objeto, dejando a las subclases decidir de qué tipo de clase se realizará la instancia en tiempo de ejecución. Reducir el uso del operador new.

Ejemplo: Usamos el patrón Factory cuando una clase no puede anticipar que clase de objetos debe crear, esto se sabe durante el tiempo de ejecución. O cuando un método regresa una de muchas posibles clases que comparten características comunes a través de una superclase. O para encapsular la creación de objetos.

Prototype

Propósito: Especificar varios tipos de objetos que pueden ser creados en un prototipo para crear nuevos objetos copiando ese prototipo. Reduce la necesidad de crear subclases.

Ejemplo: Usamos el patrón Prototype cuando queremos crear nuevos objetos mediante la clonación o copia de otros. O cuando tenemos muchas clases potenciales que queremos usar sólo si son requeridas durante el tiempo de ejecución.

Abstract Factory

Es una jerarquía que encapsula muchas familias posibles y la creación de un conjunto de productos. El objeto "fábrica" tiene la responsabilidad de proporcionar servicios de creación para toda una familia de productos. Los "clientes" nunca crean directamente los objetos de la familia, piden la fábrica que los cree por ellos.

Ejemplo: Usamos el patrón Cuando tenemos una o múltiples familias de productos. O cuando tenemos muchos objetos que pueden ser cambiados o agregados durante el tiempo de ejecución.

Marco Práctico:

1. Patrón de Diseño Abstract Factory

Abstract Factory es un patrón de diseño creacional, que te permite producir familias de objetos relacionados con sus clases concretas. Para hacer más simple la explicación tomaremos el siguiente ejemplo.

Problema

Imagina que estás creando un simulador de una tienda de muebles. Tu código consiste en las clases que representan lo siguiente:

1. Una familia de productos relacionados tal como: Chair + Sofa + CoffeTable.
2. La colección de objetos puede tener varias combinaciones, por ejemplo: Chair + Sofa + CoffeTable que están disponibles en sus diferentes variaciones: Modern, Victorian, ArcDeco.

Necesitas una forma de crear los objetos muebles individuales de tal forma que sea posible agruparse con los otros objetos de su misma familia, a razón es porque los clientes se enojan cuando no reciben una agrupación de muebles cuando lo requieren.

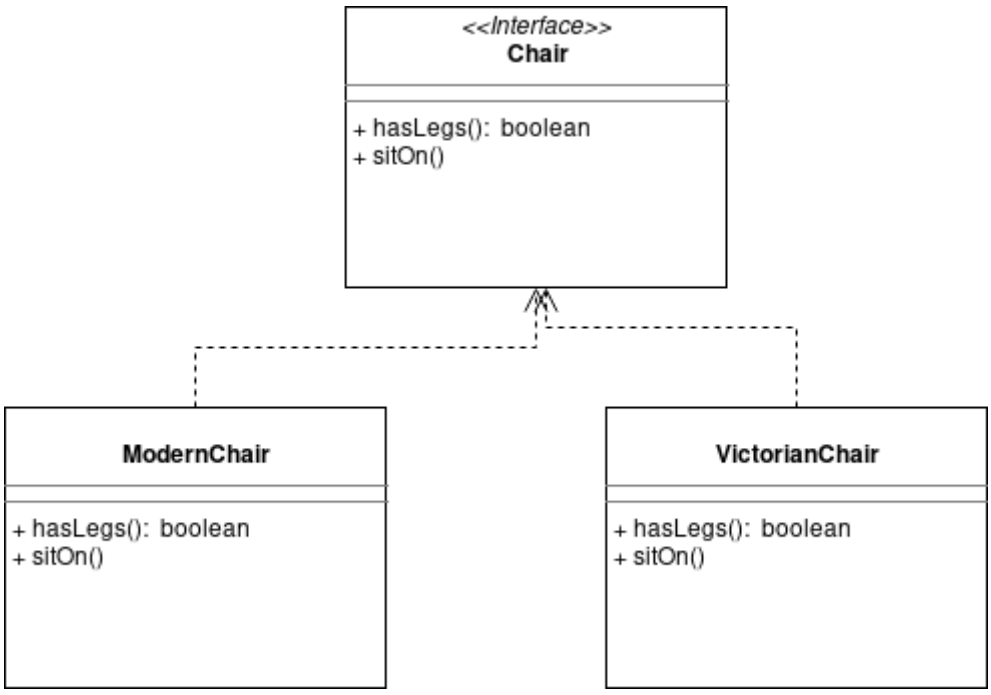
Considerando que tu no quieres cambiar el código existente cada que se añaden nuevos productos o familia de productos al programa y por su puesto esto no es para nada una buena práctica. Los vendedores de la mueblería actualizan los catálogos con mucha frecuencia y tu no debes hacer cambios al código cada vez que se esto sucede.

Implementar la Solución

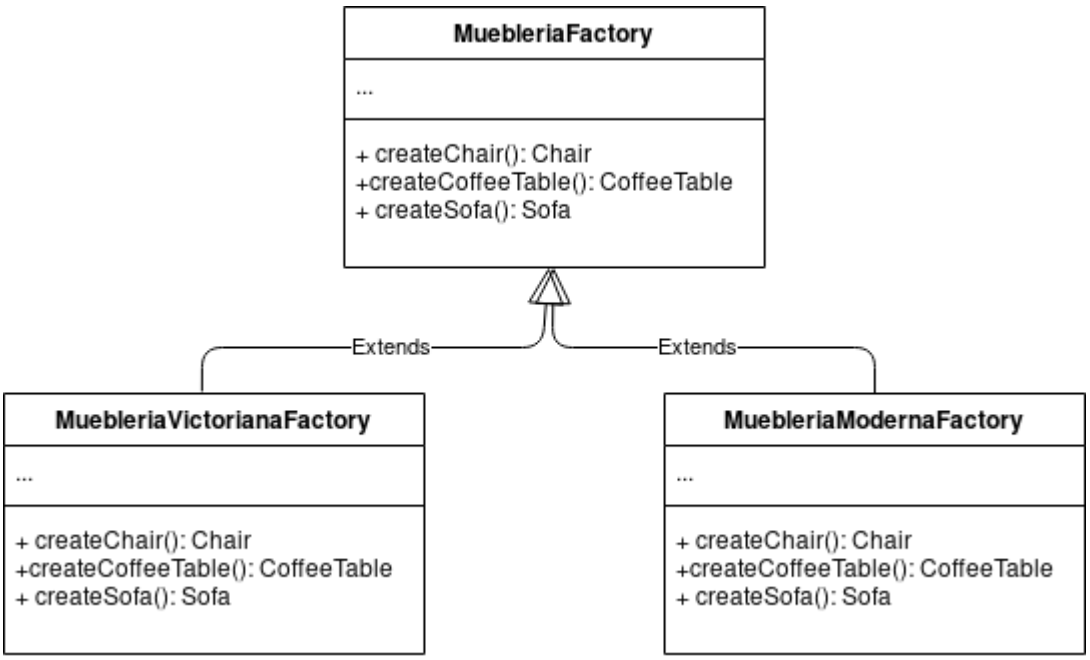
1. Se Solicita Diseñar una Aplicación en C++ que de respuesta a este problema.

Material Xtra.

Lo primero que el patrón Abstract Factory sugiere es que explicitamente las interfaces se declaren por cada producto distinto de la familia del producto. Después puedes hacer las variantes de los productos siguiendo estás interfaces. Por ejemplo, todas las variantes que pueden implementar la interface Chair

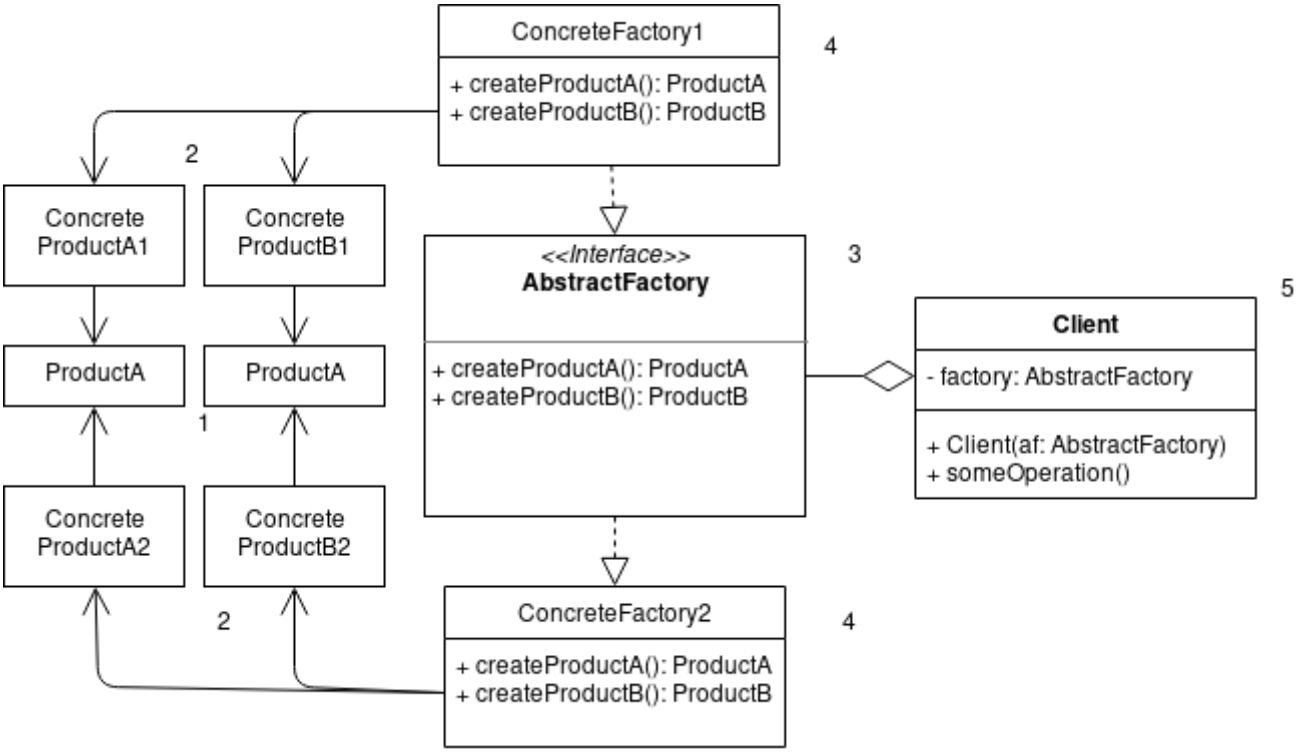


En el siguiente diagrama podrás ver la interfaz Abstract Factory, una interfaz con una lista de metodos para todos los productos que son parte de la familia de productos. Estos metodos regresan un tipo de producto abstracto representado por las interfaces que se han mostrado previamente: Chair, Sofa, etc.



Por cada variante de una familia de productos, creamos una clase fabrica separada basada en la interfaz AbstractFactory. Una Fabrica es una clase que regresa productos de un tipo particular. por ejemplo, MuebleriamodernaFactory puede crear objetos de tipo ModernChair, ModernSofa y ModernCoffeeTable.

Estructura base



1. **Abstract Products declara interfaces para un conjunto de productos distintos pero relacionados cada cual con su familia.**

2. **Concrete Products: son varias implementaciones de productos abstractos, agrupados por variantes. Cada producto abstracto debe ser implementado en todas las variantes dadas (Victorian/Modern).**
3. **Abstract Factory: Interfaz que declara un conjunto de métodos para crear cada uno de los productos abstractos.**
4. **Concrete Factories: Implementa la creación de métodos de abstract factory. Cada fabrica concreta corresponde a una variante especifica de productos y crea solo esas variantes de productos.**

Aplicaciones

1. **Usa la Abstract Factory cuando tú código necesite trabajar con varias falimias de productos, pero no quieras que dependa de las clases concretas de estos productos, puedes ser desconocidos de antemano o simplemente permitir extendibilidad furura.**

La Cátedra.
Lic. Oemig José Luis.