**“Año de la consolidación del Mar de Grau”**

**FACULTAD DE CIENCIAS E INGENIERÍA**

**E.A.P. INGENIERÍA DE SISTEMAS E INFORMÁTICA**

**TEMA**

**“PATRONES FACTORY”**

**Curso**

**Ingeniería de Software III**

**Ciclo**

**VII**

**Profesor**

CORONEL CASTILLO ERIC GUSTAVO

**PRESENTADO POR**

1. CALDAS MORENO, LEONARDO
2. ESCAJADILLO CHAVEZ, ASTRID
3. Muñoz ROSALES, MICCHELLY
4. UCHUYA GALLAC, PABLO

Los Olivos, septiembre 2016

Índice

[**Introducción** 3](#_Toc463307326)

[**Definición** 4](#_Toc463307327)

[**Elementos básicos:** 5](#_Toc463307328)

**[Características](#_Toc463307331)** [8](#_Toc463307331)

[**Código de un Factory simple JS** 9](#_Toc463307332)

[**Marco Teórico** 10](#_Toc463307333)

[**Patrón Factory:** 10](#_Toc463307334)

[**Tipos de Patrones:** 12](#_Toc463307335)

[**Relación entre los Patrones Factory:** 12](#_Toc463307336)

[**PATRÓN ABSTRACT FACTORY** 12](#_Toc463307337)

[**Aplicabilidad:** 13](#_Toc463307338)

[**FACTORY METHOD:** 13](#_Toc463307339)

[**Objetivos de Factory Method** 13](#_Toc463307340)

[**FACTORY SIMPLE (FACTORY PATTERN):** 14](#_Toc463307341)

[**Diferencias entre Factory Method y Abstract Factory** 15](#_Toc463307342)

[**VENTAJAS:** 17](#_Toc463307343)

[**DESVENTAJAS:** 18](#_Toc463307344)

[**RECOMENDACIONES:** 18](#_Toc463307345)

[**CONCLUSIONES:** 19](#_Toc463307346)

[**Bibliografías** 20](#_Toc463307347)

**PATRONES FACTORY**

# **Introducción**

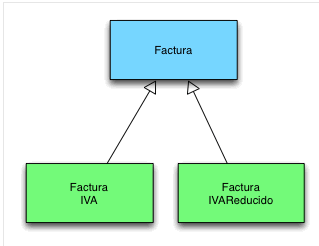
Los patrones como elemento de la reutilización, comenzaron a utilizarse en la arquitectura con el objetivo de reutilizar diseños que se habían aplicado en otras construcciones y que se catalogaron como completos. Christopher Alexander fue el primero en intentar crear un formato específico para patrones en la arquitectura. Alexander argumenta que los métodos comunes aplicados en la arquitectura dan lugar a productos que no satisfacen las demandas y requerimientos de los usuarios y son ineficientes a la hora de conseguir el propósito de todo diseño y esfuerzo de la ingeniería: mejorar la condición humana. Alexander describe algunos diseños eternos para tratar de conseguir sus metas. Propone, así, un paradigma para la arquitectura basado en tres conceptos: la calidad, la puerta y el camino.

Es un tema importante en el desarrollo de software actual ya que permite capturar la experiencia. Busca ayudar a la comunidad de desarrolladores de software a resolver problemas comunes, creando un cuerpo literario de base Crea un lenguaje común para comunicar ideas y experiencia acerca de los problemas y sus soluciones   
El uso de patrones ayuda a obtener un software de calidad (reutilización y extensibilidad)

# **Definición**

Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y describe también el núcleo de la solución al problema, de forma que puede utilizarse un millón de veces sin tener que hacer dos veces lo mismo.

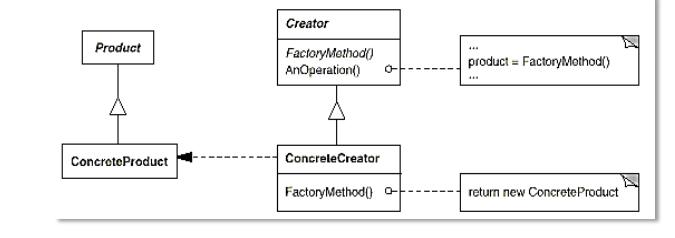
Uno de los patrones de diseño más utilizados en Java es el patrón Factory que es un patrón de diseño creacional y que sirve para construir una jerarquía de clases. Sin embargo, a veces a la gente le cuesta ver cómo usar este patrón en su código. Vamos a utilizar un ejemplo sencillo en el que tendremos una jerarquía de clases factura como se muestra a continuación.



# **Elementos básicos:**

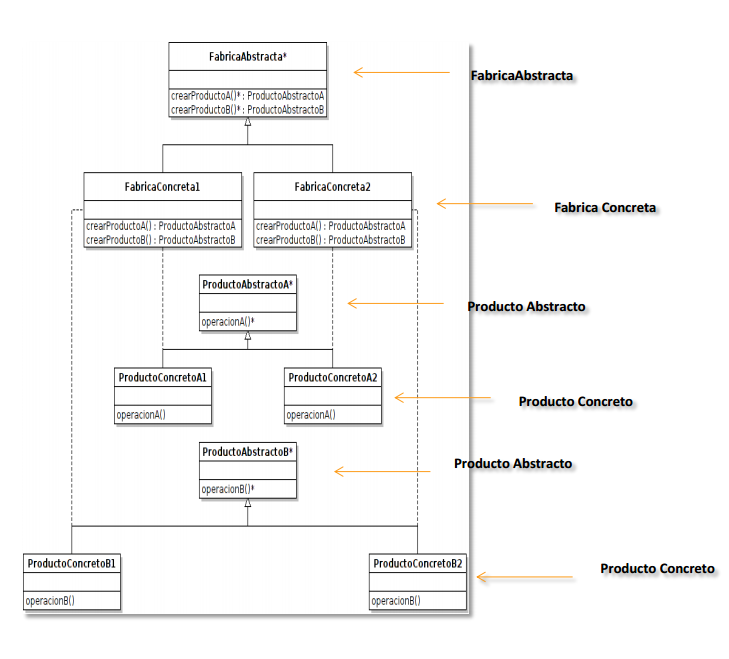
### **Elementos del Factory Method:**

* Producto o Product: Define la interfaz de los objetos creados por el método de fabricación.
* ProductoConcreto o ConcreteProduct: Implementa la interfaz de los productos
* Creador o Creator o Factoría o Factory: Declara el método de fabricación y opcionalmente pude de definir una implementación por defecto que construye un producto concreto. Puede utilizar el método de fabricación.
* CreadorConcreto o ConcreteCreator o FactoríaConcreta o ConcreteFactory: Redefine el método de fabricación para devolver una instancia de un producto concreto.



### **Elementos del Abstract Factory :**

* Fabrica Abstracta: Define un conjunto de métodos (interfaz) para la creación de productos abstractos. Declara una interfaz para operaciones que crean objetos producto abstracto.
* Fabrica Concreta1/2: Implementa la interfaz de la FabricaAbstracta para la creación de los distintos productos concretos. : Implementa las operaciones para crear productos de objetos concretos
* Producto Abstracto A\*/B\*: Define la interfaz de los objetos de tipo ProductoA/B. Declara la interfaz para un tipo de objeto producto
* ProductoConcretoA1/A2/B1/B2: Implementan su respectiva interfaz representando un producto concreto. Define un objeto producto que será creado por una fábrica concreta correspondiente. Implementa interfaz de producto abstracto
* Cliente: Sólo usa interfaces declaradas por las clases Fábrica Abstracta y Producto Abstracto.



# **Características**

**Patrón FACTORY METHOD**

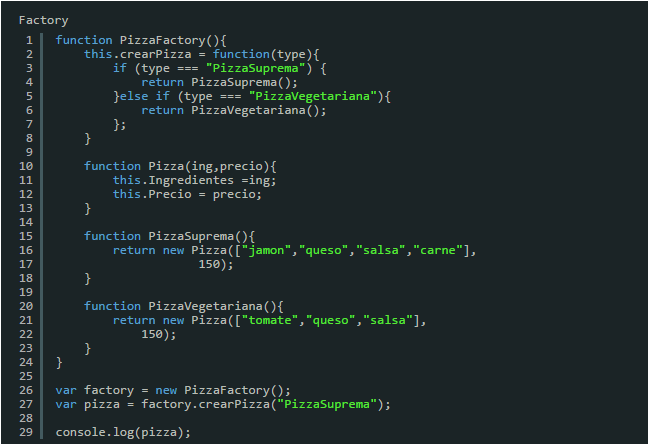
* Permite escribir aplicaciones que son más flexibles respecto de los tipos a utilizar difiriendo la creación de las instancias en el sistema a subclases que pueden ser extendidas a medida que evoluciona el sistema
* Permite encapsular el conocimiento referente a la creación de objetos.
* También hace que el diseño sea más adaptable a cambio de solo un poco más de complejidad
* Se obliga al cliente a definir subclases “Creador”, solo para crear un producto concreto y esto no puede ser apropiado siempre.

**Patrón FACTORY ABSTRACT**

* El programa maneja un tipo de objetos con unas características comunes y algunas propias. En general se resuelve mediante el uso de dos características comunes en los lenguajes de programación orientado a objetos, las clases abstractas y los interfaces.
* Aborda el problema de la creación de familias de objetos que comparten toda una serie de características comunes en los objetos que componen dicha familias.
* Se recomienda para situaciones en las que tenemos una familia de productos concretos y prevemos la inclusión de distintas familias de productos en un futuro.

# **Código de un Factory simple JS**

Este código permite crear pizzas, entonces dependiendo del tipo de pizza que se solicite al método*crearPizza*, este método crear una pizza suprema o una pizza vegetariana.



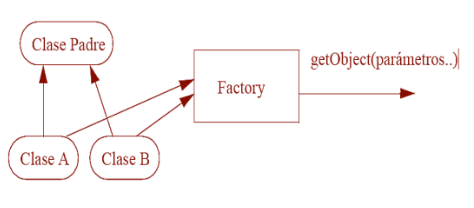
Al finalizar la ejecución del código anterior obtendremos un objeto de tipo pizza suprema, pero sin importar que tipo de pizza ordenemos el objeto contara con las propiedades *Ingredientes* y *Precio*.

# **Marco Teórico**

## **Patrón Factory:**

Factory es un patrón de creación, es decir, un objeto que maneja la creación de otros objetos. Se utilizan cuando la creación de un objeto implica más que una simple instanciación.

Este tipo de patrón se usa bastante debido a su utilidad. Su objetivo es devolver una instancia de múltiples tipos de objetos, normalmente todos estos objetos provienen de una misma clase padre mientras que se diferencian entre ellos por algún aspecto de comportamiento. Consta de un Clase que según los parámetros de creación puede crear un objeto u otro. El objeto Factory será el encargado de decidir según los parámetros que le pasemos el tipo de objeto que nos devolverá. Ejemplo del uso de patrones Factory:



En el ejemplo tenemos una clase Factory que nos devolverá un tipo de vehículo determinado según el parámetro que le pasemos y en base a constantes definidas en la propia clase.

public classVehiculo {

double velocidad; double peso;

.......

}

// Clase padre

public class Camion extends Vehiculo {

String tipoMercancia;

.....

}

// primera clase hija

public class Coche extends Vehiculo {

String asientos;

}

// segunda clase hija

public classVehiculoFactory {

....

public Vehiculo getVehiculo(int tipo) {

if (tipo == VehiculoFactory.CAMION ) { return new Camion();

} else {

return new Coche();

}

}

}

### **Tipos de Patrones:**

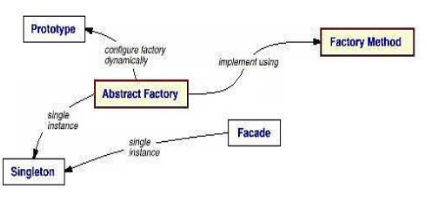
Factory Method

Abstract Factory

### **Relación entre los Patrones Factory:**

Los patrones no existen aislados el uno del otro, sino más bien dentro del contexto de un lenguaje o sistema de patrones. Por consiguiente existen relaciones entre ellos que pueden determinar cuándo, cómo y por qué utilizar uno u otro.

El patrón Abstract Factory se implementa utilizando Factory Method. A su vez, ambos patrones están relacionados con otros patrones del catálogo.



## **PATRÓN ABSTRACT FACTORY**

El patrón Abstract Factory proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas. Dado un conjunto de clases abstractas relacionadas, permite el modo de crear instancias de estas clases abstractas desde el correspondiente conjunto de subclases concretas.

### **Aplicabilidad:**

Este patrón se puede aplicar cuando:

* Un sistema debe ser independiente de cómo sus objetos son creados.
* Un sistema debe ser 'configurado' con una cierta familia de productos.
* Cuando es necesario hacer cumplir una restricción de que una familia de objetos sea usado conjuntamente.

## **FACTORY METHOD:**

El patrón de diseño Factory Method consiste en utilizar una clase constructora abstracta con unos cuantos métodos definidos y otros abstractos: está dedicado a la construcción de objetos de un subtipo de un tipo determinado.

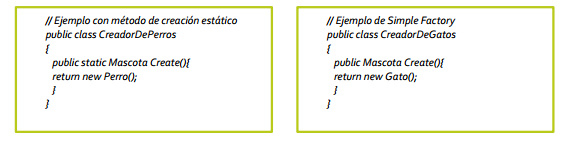
### **Objetivos de Factory Method**

* Permitir a una clase diferir la instanciación a subclases: Aplicaciones más flexibles que definen una interfaz para crear un objeto, pero dejando que las subclases decidan cuál clase instanciar. Estas subclases pueden ser extendidas a medida que evoluciona el sistema.
* La definición de un constructor "virtual".
* Reducir el uso del operador new.

## **FACTORY SIMPLE (FACTORY PATTERN):**

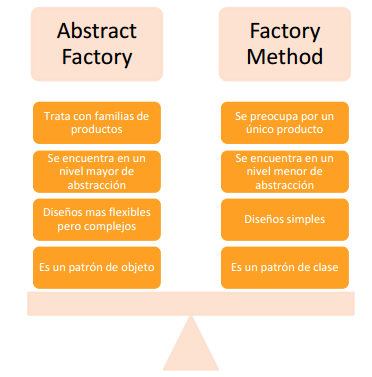
El Factory Simple son clases con métodos estáticos de fabricación o fábricas concretas que tienen implementación, pero sus métodos no son redefinibles.

En estos casos, estamos ante una implementación del patrón Simple Factory. Hemos dejado este tipo de factorías para el final, dado que son más fáciles de entender y analizar que los otros patrones presentados anteriormente. En este caso, estamos ante tipos que tienen la responsabilidad de crear instancias de objetos de otras clases, pero que no cumplen con las premisas establecidas en los patrones anteriores.



En el primer caso, el método de creación es estático. Note que no pueden redefinirse los métodos de fabricación en ninguna de las dos clases.

## **Diferencias entre Factory Method y Abstract Factory**

* Abstract Factory generalmente se implementa utilizando Factory Method y por tanto provee al menos toda la flexibilidad de éste. La diferencia principal entre ambos es que el primero trata con familias de productos, mientras que el otro se preocupa por un único producto.
* Abstract Factory se encuentra a un nivel de abstracción mayor que Factory Method.
* Los diseños que usan Abstract Factory son más flexibles que los que utilizan Factory Method, pero son también más complejos.
*  Otra diferencia notable es el ámbito de ambos patrones: Factory Method es un patrón de clase, mientras que Abstract Factory es un patrón de objeto. Los patrones de clase se refieren a las relaciones entre las clases (estáticas, en tiempo de compilación) y sus subclases mientras que los de objetos tratan sobre relaciones entre instancias (dinámicas, en tiempo de ejecución

**Diferencias entre Simple Factory y Factory Method**

Comencemos con Factory Method. Para ello es fundamental recordar la intención de este patrón: Factory Method define una interfaz para crear objetos, pero deja que sean las subclases las que decidan qué clases instanciar. Permite que una clase delegue en sus subclases la creación de objetos. En los ejemplos presentados en el Código Prueba, las clases no exponen una interfaz que pueda ser redefinida por clases que deriven de ésta. Por tanto, no pueden dejar que sean las subclases las que decidan qué clases instanciar ni delegar en las subclases la creación de objetos. En ambos casos esto se debe a que el comportamiento está definido en cada clase y no puede ser redefinido en sus clases derivadas.

**Diferencias entre Simple Factory y Abstract Factory**

Continuemos con Abstract Factory. De la misma forma que en el caso anterior, recordaremos la intención de este patrón: Proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas. En los ejemplos presentados en el CódigoPrueba, no exponen una interfaz para crear familias de objetos relacionados, dado que solo crean un único producto y tampoco pueden ser redefinidos por las clases derivadas.

# **VENTAJAS:**

**Patrón FACTORY METHOD**

* Se gana en flexibilidad, pues crear los objetos dentro de una clase con un "Método de Fábrica" es siempre más flexible que hacerlo directamente, debido a que se elimina la necesidad de atar nuestra aplicación unas clases de productos concretos.
* Se facilitan futuras ampliaciones, puesto que se ofrece las subclases la posibilidad de proporcionar una versión extendida de un objeto, con sólo aplicar en los Productos la misma idea del "Método de Fábrica".
* Se facilita, en cuanto a que se hace natural, la conexión entre jerarquías de clases paralelas, que son aquellas que se generan cuando una clase delega algunas de sus responsabilidades en una clase aparte. Ambas jerarquías de clases paralelas son creadas por un mismo cliente y el patrón Método de Fábrica establece la relación entre parejas de subclases concretas en las mismas.
* Centralización de la creación de objetos
* Facilita la escalabilidad del sistema
* El usuario se abstrae de la instancia a crear.
* En general, la gran ventaja del uso de patrones es que minimizan el riesgo de generar un mal diseño y permiten comunicar experiencias entre diseñadores

**Patrón FACTORY ABSTRACT**

* proporciona una interfaz para crear familias de objetos que dependen entre sí.
* Flexibilidad al aislar las clases concreta.
* Facilita el intercambio de familias de productos sin mezclarse.
* Fomenta la consistencia de productos.

# **DESVENTAJAS:**

**Patrón FACTORY METHOD**

* Se obliga al cliente a definir subclases de la clase "Creador" sólo para crear un producto concreto y esto puede no ser apropiado siempre.

**Patrón FACTORY ABSTRACT**

* Agregar nuevos productos es complicado ya que se deben modificar las Abstract Factory y Concret Factory.

# **RECOMENDACIONES:**

**Patrón FACTORY METHOD**

* Una clase no puede prever la clase de objetos que tiene que crear.
* Una clase quiere que sus subclases decidan qué objetos crean.
* Las clases delegan responsabilidades a una de entre varias subclases auxiliares, y queremos localizar en qué subclase concreta se ha delegado.

**Patrón FACTORY ABSTRACT**

* Un sistema debe ser independiente de cómo se crean, componen y representan sus productos.
* Un sistema debe configurarse con una de entre varias familias de productos.
* Una familia de productos relacionados están hechos para usarse juntos, y se necesita cumplir esa restricción.
* Se desea ofrecer una biblioteca de clases-producto, revelando sus interfaces pero no sus implementaciones.

# **CONCLUSIONES:**

* Los patrones de diseño ayudan a diseñar más rápido.
* Describen la solución a problemas que se repiten una y otra vez en nuestros sistemas, de forma que se puede usar esa solución siempre que haga falta.
* Capturan el conocimiento que tienen los expertos a la hora de diseñar.
* Ayudan a generar software “maleable” (software que soporta y facilita el cambio, la reutilización y la mejora).
* Son guías de diseño, no reglas rigurosas.

# **Bibliografías**

<http://informaticapc.com/patrones-de-diseno/factory-method.php>

<https://msdn.microsoft.com/es-es/library/bb972258.aspx>

<https://msdn.microsoft.com/es-es/library/bb972258.aspx#M12>

<http://docplayer.es/17502373-Factory-method-gamma-et-al.html>

<https://prezi.com/-xr8qji5pvco/factory-method/>

<https://msdn.microsoft.com/es-mx/library/bb972258.aspx>

<http://www.7sabores.com/blog/implementar-patron-diseno-factory-javascript>