**Abstract Factory**

Este patrón crea diferentes familias de objetos. Su objetivo principal es soportar múltiples estándares que vienen definidos por las diferentes jerarquías de herencia de objetos. Es similar al Factory Method, sólo que esta orientado a combinar productos.

Se debe utilizar este patrón cuando:

* Un sistema se debe configurar con una de entre varias familias de productos.
* Una familia de productos que están relacionados y fueron hechos para utilizarse juntos.

Diagrama de Clases

Diagrama

Descripción generada automáticamente

AbstractFactory: declara una interfaz para la creación de objetos de productos abstractos.

ConcreteFactory: implementa las operaciones para la creación de objetos de productos concretos.

AbstractProduct: declara una interfaz para los objetos de un tipo de productos.

ConcreteProduct: define un objeto de producto que la correspondiente factoría concreta se encargaría de crear, a la vez que implementa la interfaz de producto abstracto.

Client: utiliza solamente las interfaces declaradas en la factoría y en los productos abstractos.

Una única instancia de cada FactoryConcreto es creada en tiempo de ejecución. AbstractFactory delega la creación de productos a sus subclases FactoryConcreto.

Ahora que explique que rol ocupa cada uno en el diagrama, les pido un poco de atención en lo siguiente: veamos que relación tienen los FactoryConcretos con respectos a los productos. Esto es, FactoryConcreto1 crea una relación entre un producto de la familia A y un producto de la familia B. Y, por otro lado, tenemos que el FactoryConcreto2 crea una relación entre otros dos productos de ambas familias.

Esto ya debería darnos una pista sobre el funcionamiento del AbstractFactory: se crea una clase por cada relación que necesitemos crear. Esto quedará más claro en el ejemplo a continuación.

Ejemplo



Hagamos de cuenta que tenemos dos familias de objetos:

1) La clase TV, que tiene dos hijas: Plasma y LCD.

2) La clase Color, que tiene dos hijas: Amarillo y Azul (los mejores colores, sin duda! ).

Más alla de todos los atributos/métodos que puedan tener la clase Color y TV, lo importante aquí es destacar que Color define un método abstracto:

public abstract void colorea(TV tv);

Este método es la relación que une las dos familias de productos.Dado que es un método abstracto, Azul debe redefinirlo:

public void colorea(TV tv) {

System.out.println("Pintando de azul el "+ tv.getDescripcion());

}

Lo mismo ocurre con Amarillo:

public void colorea(TV tv) {

System.out.println("Pintando de amarillo el "+ tv.getDescripcion());

}

Bien, veamos las clases correspondientes antes de continuar con nuestros ejemplo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Escenario: nuestra empresa se dedica a darle un formato estético específico a los televisores LCD y Plasma. Se ha decidido que todos los LCD que saldrán al mercado serán azules y los plasma serán amarillos. Ahora bien, una solución simple sería en la clase Azul colocar el LCD y en la clase Amarillo colocar el Plasma y todo funcionaría de maravillas. ¿Cual sería el problema? Que esta todo hardcodeado. Esto quiere decir que el hecho de que los LCD sean azules y los plasmas amarillos es una decisión del negocio y, como tal, puede variar (y de hecho el negocio varía constantemente).

Por ejemplo, que pasa si mañana Se agrega otro color o me cambian el color del LCD o mucho peor, ¿que pasa si se crea otro producto LED y también se lo quiere pintar de Azul?

Para evitar un dolor de cabeza conviene separar estas familias y utilizar el Abstract Factory:

Texto

Descripción generada automáticamente

Y los FactoryConcretos, que relacionan las familias:

Texto, Carta

Descripción generada automáticamente

Texto, Carta

Descripción generada automáticamente

Consecuencias

Se oculta a los clientes las clases de implementación: los clientes manipulan los objetos a través de las interfaces o clases abstractas.

Facilita el intercambio de familias de productos: al crear una familia completa de objetos con una factoría abstracta, es fácil cambiar toda la familia de una vez simplemente cambiando la factoría concreta.

Mejora la consistencia entre productos: el uso de la factoría abstracta permite forzar a utilizar un conjunto de objetos de una misma familia.

Como inconveniente podemos decir que no siempre es fácil soportar nuevos tipos de productos si se tiene que extender la interfaz de la Factoría abstracta.