Abstract Factory Pattern

Introducción y nombre

Abstract Factory. Creacional.

Intención

Ofrece una interfaz para la creación de familias de productos relacionados o dependientes sin especificar las clases concretas a las que pertenecen.

También conocido como

Factoría Abstracta, Kit.

Motivación

El problema que intenta solucionar este patrón es el de crear diferentes familias de objetos. Su objetivo principal es soportar múltiples estándares que vienen definidos por las diferentes familias de objetos. Es similar al Factory Method, sólo le añade mayor abstracción.

Solución

Se debe utilizar este patrón cuando:

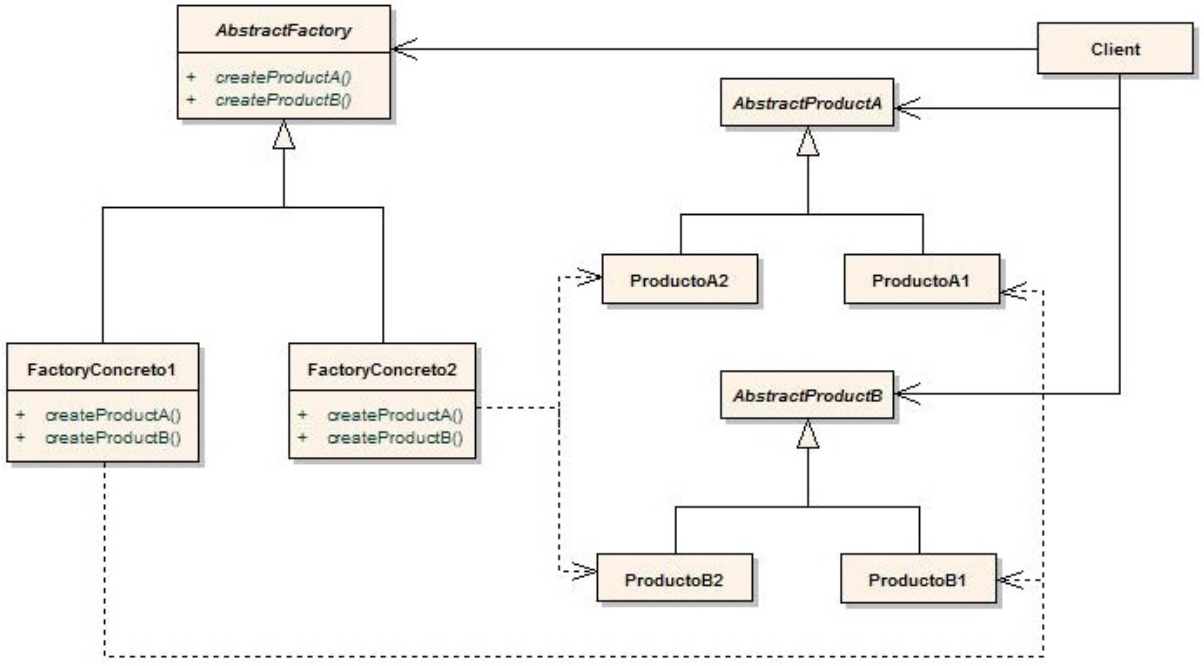
La aplicación debe ser independiente de como se crean, se componen y se representan sus productos.

Un sistema se debe configurar con una de entre varias familias de productos.

Una familia de productos relacionados están hechos para utilizarse juntos (hay que hacer que esto se cumpla).

Para ofrecer una librería de clases, mostrando sólo sus interfaces y no sus implementaciones.

Diagrama UML



Participantes

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.

Colaboraciones

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.

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.

Implementación

Veamos los puntos más importantes para su implementación:

Factorias como singletons: lo ideal es que exista una instancia de FactoryConcreto por familia de productos.

Definir factorías extensibles: añadiendo un parámetro en las operaciones de creación que indique el tipo de objeto a crear.

Para crear los productos se usa un Factory Method para cada uno de ellos.

Código de muestra

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.

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:

[code]

```
public abstract void colorea(TV tv);

public abstract class TV {
    public abstract String getDescripcion();
}

public abstract class Color {
    public abstract void colorea(TV tv);
}
```

```
public class Amarillo extends Color {
    @Override
    public void colorea(TV tv) {
        System.out.println("Pintando de amarillo en el televisor " + tv.getDescripcion());
    }
}

public class Azul extends Color {
    @Override
    public void colorea(TV tv) {
        System.out.println("Pintando de azul en el televisor " + tv.getDescripcion());
    }
}

public class Plasma extends TV {
    @Override
    public String getDescripcion() {
        return "Plasma";
    }
}
public class LCD extends TV {
    @Override
    public String getDescripcion() {
        return "LCD";
    }
}
```

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. Por este motivo se ha decidido realizar el patrón Abstract Factory:

```
public abstract class AbstractFactory {
    public abstract TV createTV();

    public abstract Color createColor();
}

public class FactoryLcdAzul extends AbstractFactory {
    @Override
    public Color createColor() {
        return new Azul();
    }

    @Override
    public TV createTV() {
        return new LCD();
    }
}

public class FactoryPlasmaAmarillo extends AbstractFactory {
    @Override
    public Color createColor() {
        return new Amarillo();
    }

    @Override
    public TV createTV() {
        return new Plasma();
    }
}
```

Y, por último, la clase Cliente de estas Factory:

```
public class EnsamblajeTV {
    public EnsamblajeTV(AbstractFactory factory) {
        Color color = factory.createColor();
        TV tv = factory.createTV();
```

```
        color.colorea(tv);
    }
}
```

Como se puede observar el código es bastante genérico. El secreto de este patrón ocurre en los Factory Concretos que es donde se definen las reglas del negocio.
Veamos esto en funcionamiento:

```
public class Main {
    public static void main(String[] args) {
        // Probando el factory LCD + Azul
        AbstractFactory f1 = new FactoryLcdAzul();
        EnsamblajeTV e1 = new EnsamblajeTV(f1);

        // Probando el factory Plasma + Amarillo
        AbstractFactory f2 = new FactoryPlasmaAmarillo();
        EnsamblajeTV e2 = new EnsamblajeTV(f2);
    }
}
```

El resultado por consola es:
Pintando de azul en el televisor LCD
Pintando de amarillo en el televisor Plasma

[/code]