Daza Corredor Alejandro
Paolo

Patrones Creacionales

Abstract Factory

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [Christopher Alexander].

1. Introducción

El concepto de patrones creacionales nos dice que los patrones de creación abstraen la forma en la que se crean los objetos, permitiendo tratar las clases a crear de forma genérica dejando para más tarde la decisión de qué clases crear o cómo crearlas, este concepto nos habla de decidir que clase concreta se instanciara en tiempo de ejecución. Uno de los patrones que nos permite implementar este concepto es el Abstract Factory, veamos de que se trata.

3. Abstract Factory

Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.

Nombre del patrón:
Abstract Factory.

Clasificación del patrón:
Creacional.

Intención:
Provee una interfaz para la creación de familias de objetos relacionados o dependientes sin especificar sus clases concretas.

También conocido como:
kit o toolkit.

Motivación:
Considere una herramienta de interfaces de usuario que soporta múltiples estándares look-and-feel, diferentes look-and-feel definen diferentes apariencias y comportamientos para las interfaces de usuario. Si una aplicación instancia clases específicas para cada elemento de un look-and-feel especifico luego sera muy difícil cambiarlo.

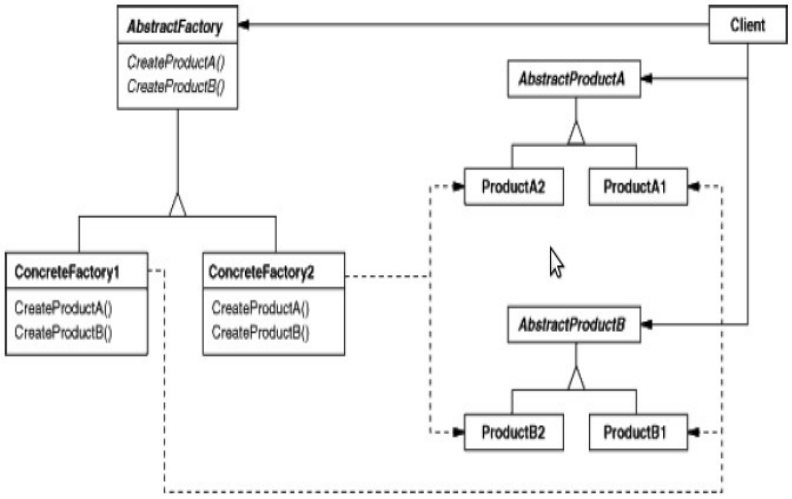
Este problema se puede solucionar al definir una fabrica abstracta que es una clase abstracta que declare una interfaz para la creación de cada uno de los elementos que pueden tener una interfaz de usuario. Teniendo también una clase abstracta para cada elemento y subclasses concretas que implementan los elementos para un look-and-feel especifico. La interfaz de la clase abstracta tiene operaciones que retornan un objeto por cada elemento de interfaz de usuario definidos en clases abstractas. Los clientes llaman estas operaciones para obtener instancias de los elementos de interfaz de usuario, pero los clientes desconocen las clases concretas que están usando y se mantienen independientes del look-and-feel actual.

Aplicabilidad:
El patrón Abstract Factory debe ser usado cuando:

- El sistema debe ser independiente de como sus productos son creados, compuestos y representados.
- El sistema debe ser configurado con uno de múltiples familias de productos.

- Una familia de objetos relacionados es diseñada para ser usada en conjunto y se debe garantizar esa restricción.
- Se desea proveer una librería de clases de productos y se desea revelar sus interfaces, no sus implementaciones.

Estructura:



Participantes:

- AbstractFactory: declara una interfaz para las operaciones que crean objetos de productos abstractos.
- ConcreteFactory: implementa las operaciones para crear productos de objetos concretos.
- AbstractProduct: declara una intefaz para un tipo de objeto de producto.
- ConcreteProduct: define un objeto de producto a ser creado por la correspondiente fabrica concreta (ConcreteFactory). Implementa la interfaz AbstractProduct.
- Client: usa solo interfaces declaradas por AbstarctFactory y AbstractProduct.

Colaboraciones:

- Normalmente una única instancia de una ConcreteFactory es creada en tiempo de ejecución, esta fabrica concreta crea objetos de productos que tienen una particular implementación. Para crear diferentes objetos de productos, los clientes deben usar diferentes fabricas concretas.
- AbstractFactory, transfiere la creación de objetos de productos a sus subclases ConcreteFactory.

Consecuencias:

- Aisla las clases concretas, ayuda a mantener un control de las clases de objetos que una aplicación crea, ya que encapsula la responsabilidad y el proceso de la creación de objetos.
- Hace fácil el intercambio de familias de productos, ya que la fabrica concreta aparece una sola vez en el código del cliente cuando esta es instanciada, esto hace fácil su cambio.
- Promueve la consistencia entre productos, cuando una familia de productos es diseñada para trabajar en conjunto, es importante que una aplicación use solo una familia de productos a la vez, este patrón hace que esta restricción sea fácil de lograr.
- Soportar nuevas clases de productos es difícil, ya que la interface AbstractFactory establece el set de productos que pueden ser creados.

Implementación:

- Fabricas como singletons, comúnmente solo se necesita una instancia de una ConcreteFactory por familia de productos, por esto es aconsejable implementarla como singleton.

- Creación de productos, AbstractFactory solo declara una interfaz para la creación de productos dejando a las subclases ConcreteFactory la creación de estos. La forma mas común de hacer esto es mediante un Factory Method por cada producto.
- Si muchas familias de productos son posibles, se debe pensar en implementar el patrón Prototype donde la fabrica concreta es inicializada con una instancia prototípica de cada producto en la familia, eliminando así la necesidad de crear una nueva clase de fabrica concreta por cada familia de productos.
- Definición de fabricas extensibles, ya sabemos que una de las desventajas del patrón es que no se puede extender la fabrica abstracta, ya que esto implicaría redefinir la interfaz del AbstractFactory y en ese sentido todas las subclases que dependan de ella, esta limitante se puede romper mediante la adición de un parámetro a las operaciones que me indique la clase de objeto que va ser creado, esto aunque hace mas flexible la fabrica es menos seguro según el manejo de tipificado del lenguaje en el que se está trabajando.

Código de ejemplo:

```
public abstract class AbstractFactory{
	public abstract AbstractProductA CreateProductA();
	public abstract AbstractProductB CreateProductB();
}

public class ConcreteFactory1 extends AbstractFactory{
	public AbstractProductA CreateProductA(){
		return new ProductA1();
	}
	public AbstractProductB CreateProductB(){
		return new ProductB1();
	}
}

class ConcreteFactory2 extends AbstractFactory{
	public AbstractProductA CreateProductA(){
		return new ProductA2();
	}
	public AbstractProductB CreateProductB(){
		return new ProductB2();
	}
}

public abstract class AbstractProductA {
}

abstract class AbstractProductB{
	public abstract void Interact(AbstractProductA a);
}

class ProductA1 extends AbstractProductA{
}

public class ProductB1 extends AbstractProductB{
	public void Interact(AbstractProductA a){
		System.out.println(this.getClass().toString() + "
			interactua con " + a.getClass().toString());
	}
}

public class ProductA2 extends AbstractProductA{
}

public class ProductB2 extends AbstractProductB{
	@Override
	public void Interact(AbstractProductA a) {
		System.out.println(this.getClass().toString() + "
			interactua con " + a.getClass().toString());
	}
}
```

```
public class Client {
	private AbstractProductA abstractProductA;
	private AbstractProductB abstractProductB;

	// Constructor
	public Client(AbstractFactory factory){
		abstractProductB = factory.CreateProductB();
		abstractProductA = factory.CreateProductA();
	}

	public void Run(){
		abstractProductB.Interact(abstractProductA);
	}
}

public class Main {
	public static void main(String[] args) {
		// Abstract factory #1
		AbstractFactory factory1 = new ConcreteFactory1();
		Client c1 = new Client(factory1);
		c1.Run();

		// Abstract factory #2
		AbstractFactory factory2 = new ConcreteFactory2();
		Client c2 = new Client(factory2);
		c2.Run();
	}
}
```

Usos conocidos:

- Los WidgetKit y DialogKit usan abstract factory para el manejo del look-and-feel.

Patrones relacionados:

- Las clases de Abstract Factory son frecuente mente implementadas usando métodos de Factory Method, pero también pueden ser implementados usando Prototype.
- Una fabrica concreta es usualmente un singleton.

Referencias bibliográficas

[1] Design Patterns. Elements of Reusable Object-Oriented Software - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Addison Wesley (GoF- Gang of Four)

[2] Patrones de Diseño, Diseño de Software Orientado a Objetos - Joaquin Garcia. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

[3] Patrones de diseño - http://es.kioskea.net/contents/genie-logiciel/design-patterns.php3.

[4] Introducción al diseño con patrones – Miguel Lagos Torres. http://www.elrincondelprogramador.com/default.asp?pag=articulos/leer.asp&id=29.

[5] Patrón de diseño Abstract Factory. http://askpipe99.blogspot.com/2007/10/patron-de-diseo-abstract-factory.html.

Daza Corredor, Alejandro Paolo
Ingeniero de Sistemas, egresado de la Universidad Distrital Francisco José de Caldas, Docente TCO del proyecto curricular de Ingeniería de Sistemas en el área de programación y estudiante de la especialización de Ingeniería de Software de la Universidad Distrital Francisco José de Caldas.

e-mail: apdaza@gmail.com