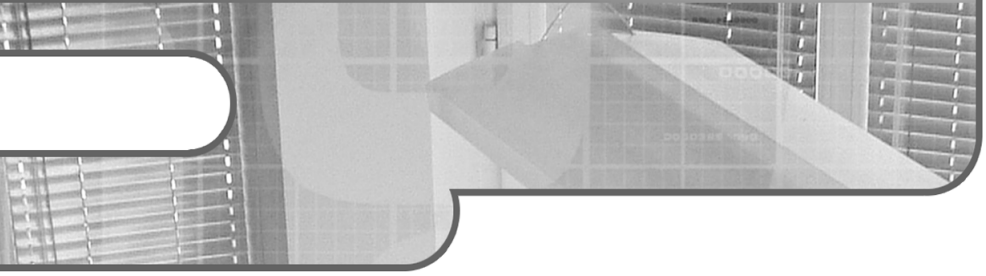

Parte 2
Patrones de construcción

Capítulo 2-1
Introducción a los patrones de construcción

1. Presentación

Los patrones de construcción tienen la vocación de abstraer los mecanismos de creación de objetos. Un sistema que utilice estos patrones se vuelve independiente de la forma en que se crean los objetos, en particular, de los mecanismos de instanciación de las clases concretas.

Estos patrones encapsulan el uso de clases concretas y favorecen así el uso de las interfaces en las relaciones entre objetos, aumentando las capacidades de abstracción en el diseño global del sistema.

De este modo el patrón `Singleton` permite construir una clase que posee una instancia como máximo. El mecanismo que gestiona el acceso a esta única instancia está encapsulado por completo en la clase, y es transparente a los clientes de la clase.

2. Problemas ligados a la creación de objetos

2.1 Problemática

En la mayoría de lenguajes orientados a objetos, la creación de objetos se realiza gracias al mecanismo de instanciación, que consiste en crear un nuevo objeto mediante la llamada al operador `new` configurado para una clase (y eventualmente los argumentos del constructor de la clase cuyo objetivo es proporcionar a los atributos su valor inicial). Tal objeto es, por consiguiente, una instancia de esta clase.

Los lenguajes de programación más utilizados a día de hoy, como Java, C++ o C#, utilizan el mecanismo del operador `new`.

En Java, una instrucción de creación de un objeto puede escribirse de la siguiente manera:

```
objeto = new Clase();
```

En ciertos casos es necesario configurar la creación de objetos. Tomemos el ejemplo de un método `construyeDoc` que crea los documentos. Puede construir documentos PDF, RTF o HTML. Generalmente el tipo de documento a crear se pasa como parámetro al método mediante una cadena de caracteres, y se obtiene el código siguiente:

```
public Documento construyeDoc(String tipoDoc)
{
  Documento resultado;

  if (tipoDoc.equals("PDF"))
    resultado = new DocumentoPDF();
  else if (tipoDoc.equals("RTF"))
    resultado = new DocumentoRTF();
  else if (tipoDoc.equals("HTML"))
    resultado = new DocumentoHTML();
  // continuación del método
}
```

Este ejemplo muestra que es difícil configurar el mecanismo de creación de objetos, la clase que se pasa como parámetro al operador `new` no puede sustituirse por una variable. El uso de instrucciones condicionales en el código del cliente a menudo resulta práctico, con el inconveniente de que un cambio en la jerarquía de las clases a instanciar implica modificaciones en el código de los clientes. En nuestro ejemplo, es necesario cambiar el código del método `construyeDoc` si se quiere agregar nuevos tipos de documento.

Observación

En lo sucesivo, ciertos lenguajes ofrecen mecanismos más o menos flexibles y a menudo bastante complejos para crear instancias a partir del nombre de una clase contenida en una variable de tipo `String`.

La dificultad es todavía mayor cuando hay que construir objetos compuestos cuyos componentes pueden instanciarse mediante clases diferentes. Por ejemplo, un conjunto de documentos puede estar formado por documentos PDF, RTF o HTML. El cliente debe conocer todas las clases posibles de los componentes y de las composiciones. Cada modificación en el conjunto de las clases se vuelve complicada de gestionar.

2.2 Soluciones propuestas por los patrones de construcción

Los patrones `Abstract Factory`, `Builder`, `Factory Method` y `Prototype` proporcionan una solución para parametrizar la creación de objetos. En el caso de los patrones `Abstract Factory`, `Builder` y `Prototype`, se utiliza un objeto como parámetro del sistema. Este objeto se encarga de realizar la instanciación de las clases. De este modo, cualquier modificación en la jerarquía de las clases sólo implica modificaciones en este objeto.

El patrón `Factory Method` proporciona una configuración básica sobre las subclases de la clase cliente. Sus subclases implementan la creación de los objetos. Cualquier cambio en la jerarquía de las clases implica, por consiguiente, una modificación de la jerarquía de las subclases de la clase cliente.

Capítulo 2-2
El patrón Abstract Factory

1. Descripción

El objetivo del patrón `Abstract Factory` es la creación de objetos agrupados en familias sin tener que conocer las clases concretas destinadas a la creación de estos objetos.

2. Ejemplo

El sistema de venta de vehículos gestiona vehículos que funcionan con gasolina y vehículos eléctricos. Esta gestión está delegada en el objeto `Catálogo` encargado de crear tales objetos.

Para cada producto, disponemos de una clase abstracta, de una subclase concreta derivando una versión del producto que funciona con gasolina y de una subclase concreta derivando una versión del producto que funciona con electricidad. Por ejemplo, en la figura 2-2.1, para el objeto Scooter, existe una clase abstracta `Scooter` y dos subclases concretas `ScooterElectricidad` y `ScooterGasolina`.

El objeto `Catálogo` puede utilizar estas subclases concretas para instanciar los productos. No obstante si fuera necesario incluir nuevas clases de familias de vehículos (diésel o mixto gasolina-eléctrico), las modificaciones a realizar en el objeto `Catálogo` pueden ser bastante pesadas.

El patrón `Abstract Factory` resuelve este problema introduciendo una interfaz `FábricaVehículo` que contiene la firma de los métodos para definir cada producto. El tipo devuelto por estos métodos está constituido por una de las clases abstractas del producto. De este modo el objeto `Catálogo` no necesita conocer las subclases concretas y permanece desacoplado de las familias de producto.

Se incluye una subclase de implementación de `FábricaVehículo` por cada familia de producto, a saber las subclases `FábricaVehículoElectricidad` y `FábricaVehículoGasolina`. Dicha subclase implementa las operaciones de creación del vehículo apropiado para la familia a la que está asociada.

El objeto `Catálogo` recibe como parámetro una instancia que responde a la interfaz `FábricaVehículo`, es decir o bien una instancia de `FábricaVehículoElectricidad`, o bien una instancia de `FábricaVehículoGasolina`. Con dicha instancia, el catálogo puede crear y manipular los vehículos sin tener que conocer las familias de vehículos y las clases concretas de instanciación correspondientes.

El conjunto de clases del patrón `Abstract Factory` para este ejemplo se detalla en la figura 2-2.1.

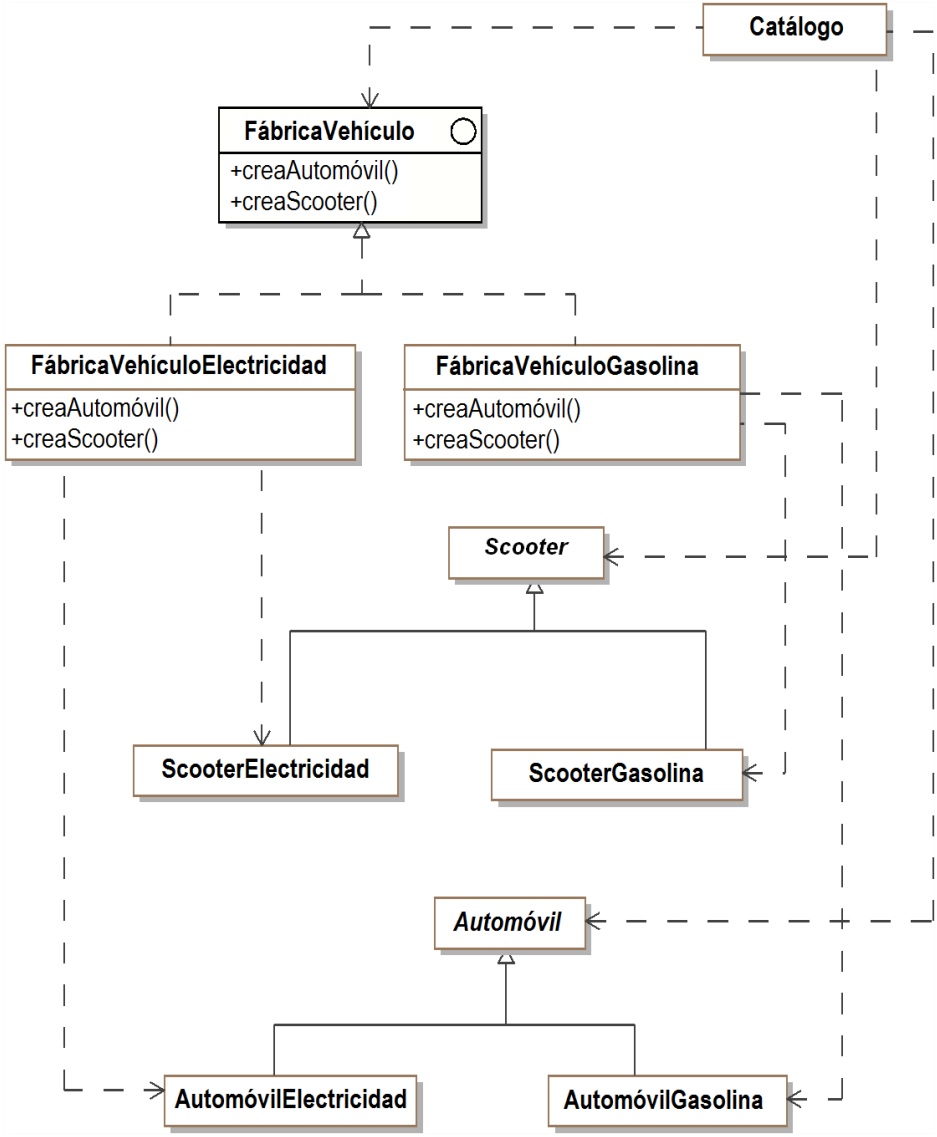

Figura 2-2.1 - El patrón `Abstract Factory` aplicado a las familias de vehículos