Patrones de construcción con Java

ABRAHAM SÁNCHEZ LÓPEZ
GRUPO MOVIS
FCC-BUAP

Introducción

- Los patrones de construcción tienen la vocación de abstraer los mecanismos de creación de objetos.
- Un sistema que utiliza 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 las 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 esta clase.

Problemas ligados a la creación de objetos, I

- En la mayoría de los lenguajes orientados a objetos, la creación de los 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).
- Los lenguajes de programación más utilizados en los últimos años, 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.
- Consideremos 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 siguiente código:

Problemas ligados a la creación de objetos, II


```
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 nos 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 quieren agregar nuevos tipos de documento.

Problemas ligados a la creación de objetos, III

- **Nota**: En lo sucesivo, ciertos lenguajes de programación 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 diferentes clases.
- 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.

Soluciones propuestas con estos patrones

- 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 de 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.

El patrón Abstract Factory, I

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.

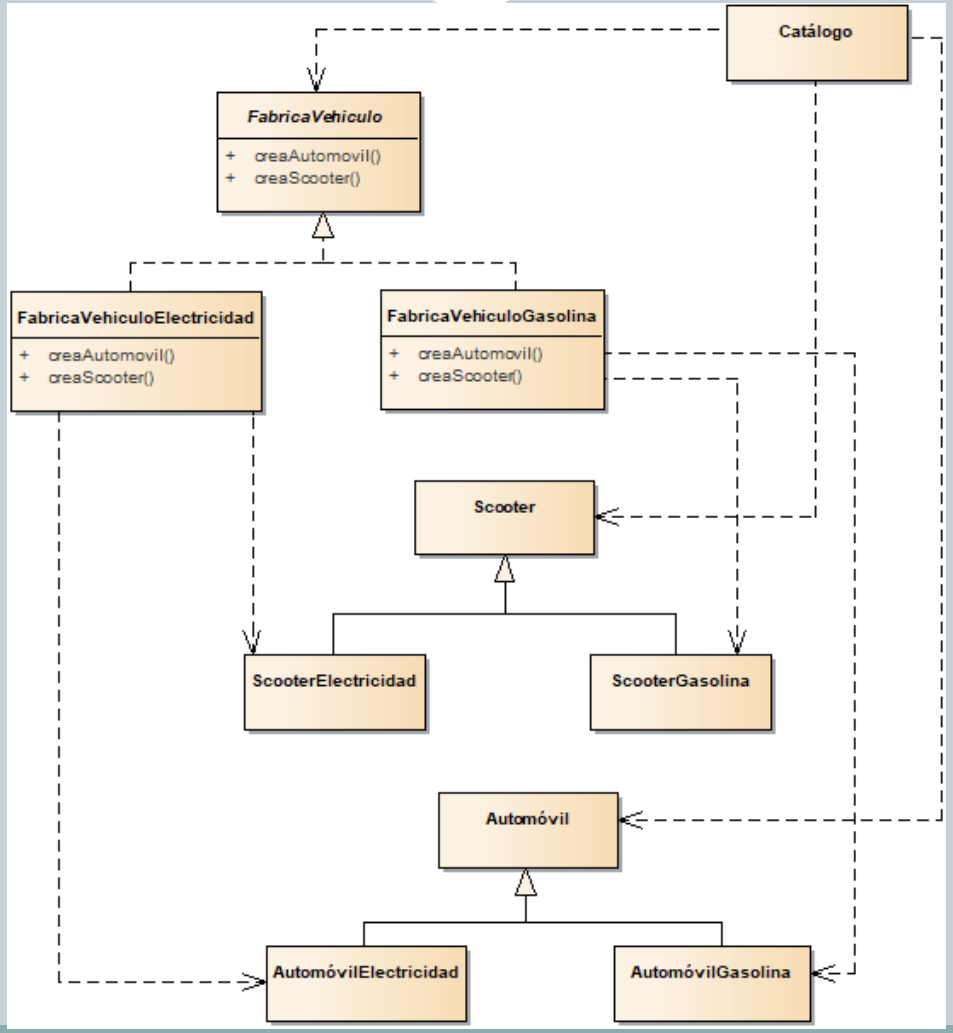
Ejemplo

- Un sistema de venta de vehículos gestiona vehículos que funcionan con gasolina y vehículos eléctricos. Esta gestión está delegada en el objeto Catalogo 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, 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 clase de familias de vehículos (diesel o mixto gasolina-eléctrico), las modificaciones a realizar en el objeto Catalogo pueden ser bastante pesadas.

El patrón Abstract Factory, II

- El patrón Abstract Factory resuelve este problema introduciendo una interfaz FabricaVehiculo 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 Catalogo no necesita conocer las subclases concretas y permanece desacoplado de las familias de producto.
- Se incluye una subclase de implementación de FabricaVehiculo por cada familia de producto, a saber las subclases FabricaVehiculoElectricidad y FabricaVehiculoGasolina.
- Dicha subclase implementa las operaciones de creación del vehículo apropiado para la familia a la que esta asociada.
- El objeto Catalogo recibe como parámetro una instancia que responde a la interfaz FabricaVehiculo, es decir o bien una instancia de FabricaVehiculoEletricidad, o bien una instancia de FabricaVehiculoGasolina.
- Con dicha instancia, el catalogo 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 muestra en la siguiente figura.

El patrón Abstract Factory



El patrón Abstract Factory, III

Estructura

1. Diagrama de clase

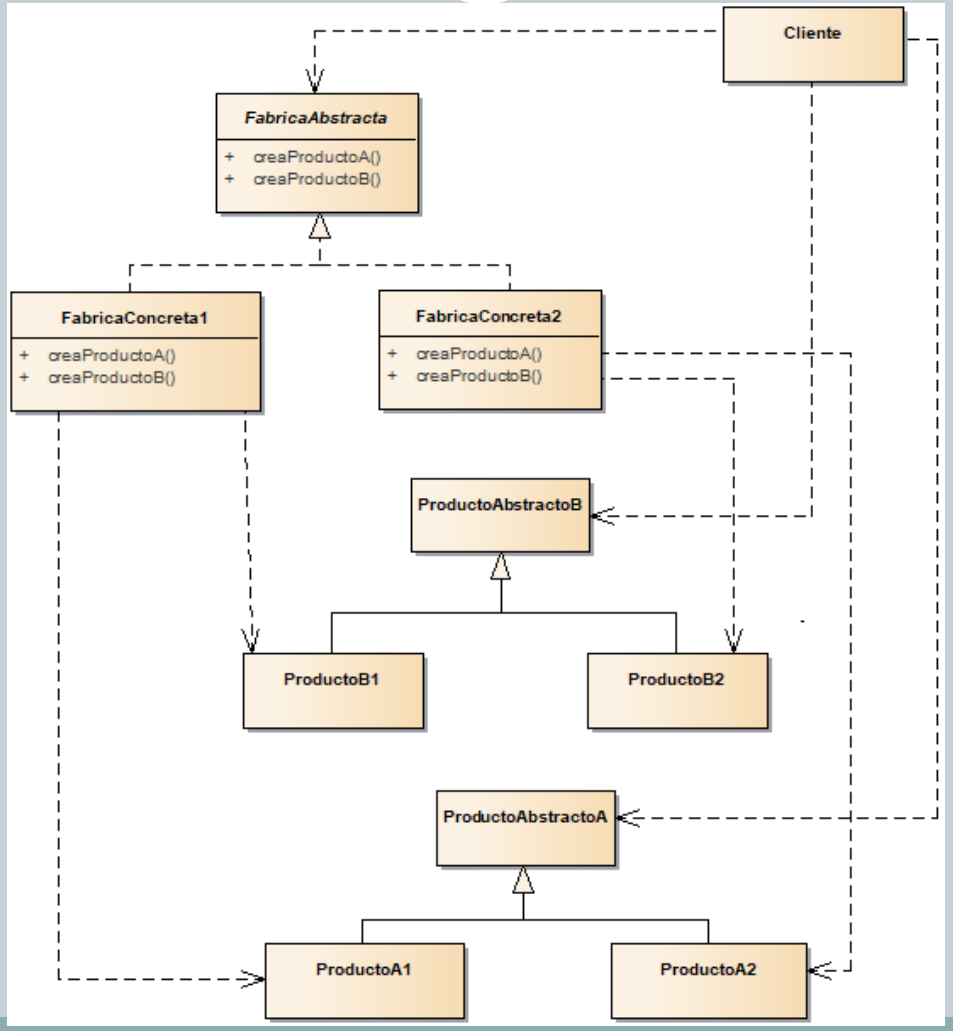
- La siguiente figura (acetato 11) detalla la estructura genérica del patrón.

2. Participantes

- Los participantes del patrón son los siguientes:
  - FabricaAbstracta (FabricaVehiculo) es una interfaz que define las firmas de los métodos que crean los distintos productos.
  - FabricaConcreta1, FabricaConcreta2 (FabricaVehiculoElectricidad, FabricaVehiculoGasolina) son las clases concretas que implementan los métodos que crean los productos para cada familia de producto. Conociendo la familia y el producto, son capaces de crear una instancia del producto para esta familia.
  - ProductoAbstractoA y ProductoAbstractoB (Scooter y Automovil) son las clases abstractas de los productos independientemente de su familia. Las familias se introducen en las subclases concretas.
  - Cliente es la clase que utiliza la interfaz FabricaAbstracta.

3. Colaboraciones

Estructura del patrón Abstract Factory



El patrón Abstract Factory, IV

- La clase Cliente utiliza una instancia de una de las fabricas concretas para crear sus productos a partir de la interfaz FabricaAbstracta.

  Nota: Normalmente sólo es necesario crear una instancia de cada fábrica concreta, que puede compartirse por varios clientes.

4. Dominios de uso

- El patrón se utiliza en los siguientes dominios:
  - Un sistema que utiliza productos necesita ser independiente de la forma en que se crean y agrupan estos productos.
  - Un sistema está configurado según varias familias de productos que pueden evolucionar.

5. Ejemplo en Java

- Presentamos a continuación un pequeño ejemplo de uso del patrón escrito en Java. El código Java correspondiente a la clase abstracta Automovil y sus subclases aparece a continuación (ver la carpeta del patrón).
- Es muy sencillo, describe los cuatro atributos de los automóviles, así como el método mostrarCaracteristicas que permite visualizarlas.

El patrón Builder, I

Descripción

- El objetivo del patrón Builder es abstraer la construcción de objetos complejos de su implementación, de modo que un cliente pueda crear objetos complejos sin tener que preocuparse de las diferencias en su implementación.

Ejemplo

- Durante la compra de un vehículo, el vendedor crea todo un conjunto de documentos que contienen en especial la solicitud de pedido y la solicitud de emplacamiento del cliente. Es posible construir estos documentos en formato HTML o en formato PDF según la elección del cliente.
- En el primer caso, el cliente le provee una instancia de la clase ConstructorDocumentacionVehiculoHTML y, en el segundo caso, una instancia de la clase ConstructorDocumentacionVehiculoPDF. El vendedor realiza a continuación, la solicitud de creación de cada documento mediante esta instancia.
- De este modo, el vendedor genera la documentación con ayuda de los métodos construyeSolicitudPedido y construyeSolicitudEmplacamiento

El patrón Builder, II

- El conjunto de clases del patrón Builder para este ejemplo se muestra en la figura del acetato 15.
- Esta figura muestra la jerarquía entre las clases ConstructorDocumentacionVehiculo y Documentacion. El vendedor puede crear las solicitudes de pedido y las solicitudes de emplacamiento sin conocer las subclases de ConstructorDocumentacionVehiculo ni las de Documentacion.
- Las relaciones de dependencia entre el cliente y las subclases de ConstructorDocumentacionVehiculo se explican por el hecho de que el cliente crea una instancia de estas subclases.

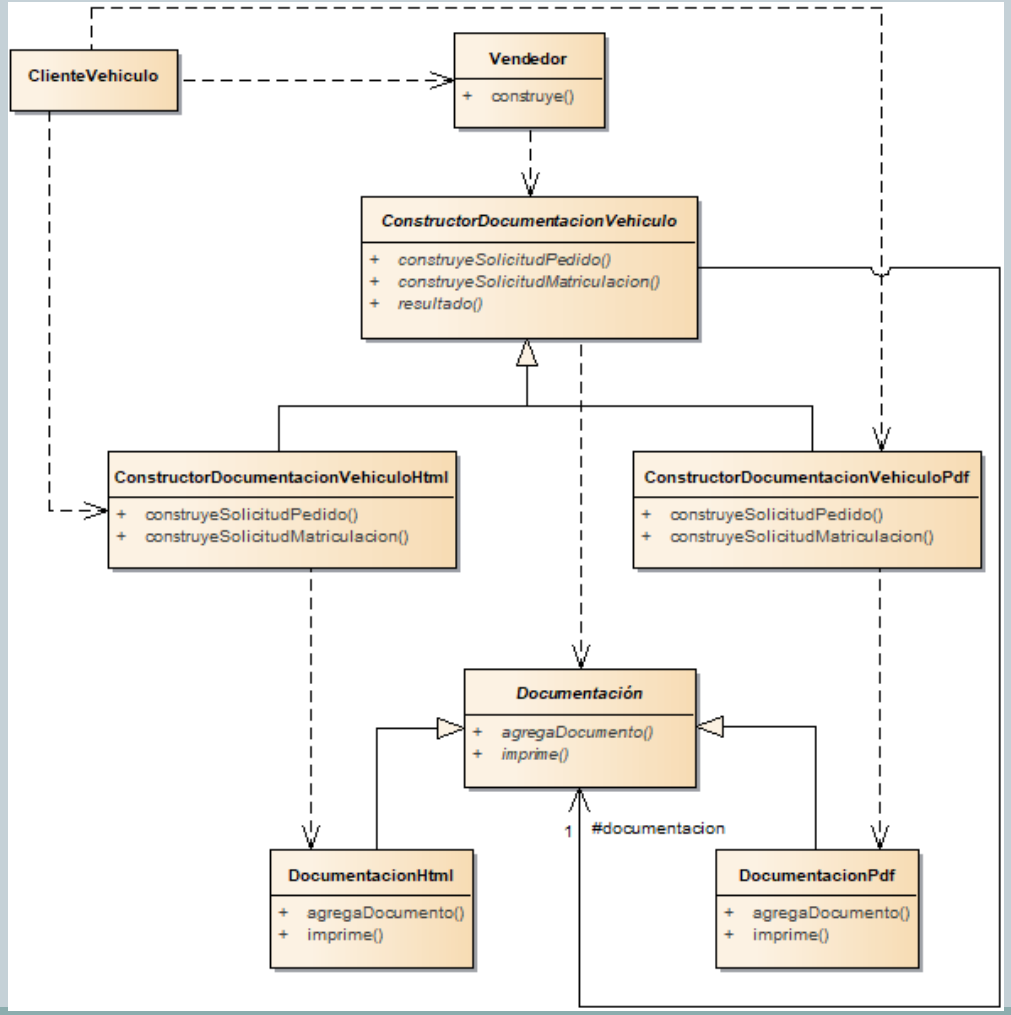
  Nota: La estructura interna de las subclases concretas de Documentacion no se muestra (entre ellas, por ejemplo, la relación de composición con la clase Documento).

Estructura

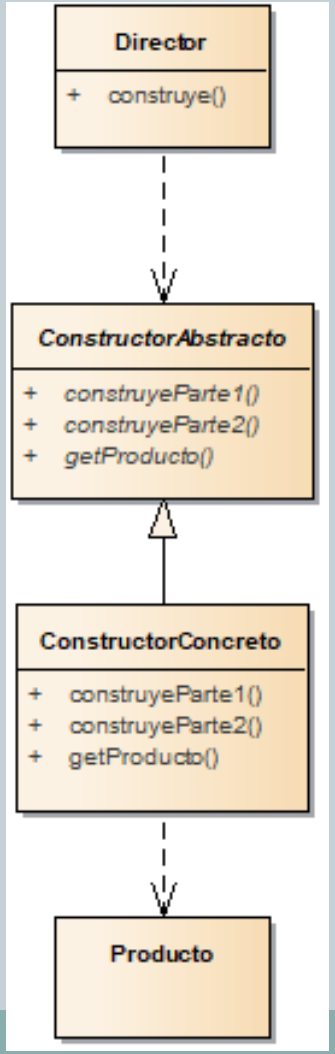
1. Diagrama de clases

- La figura del acetato 16 detalla la estructura genérica del patrón.

El patrón Builder



Estructura del patrón Builder



El patrón Builder, III

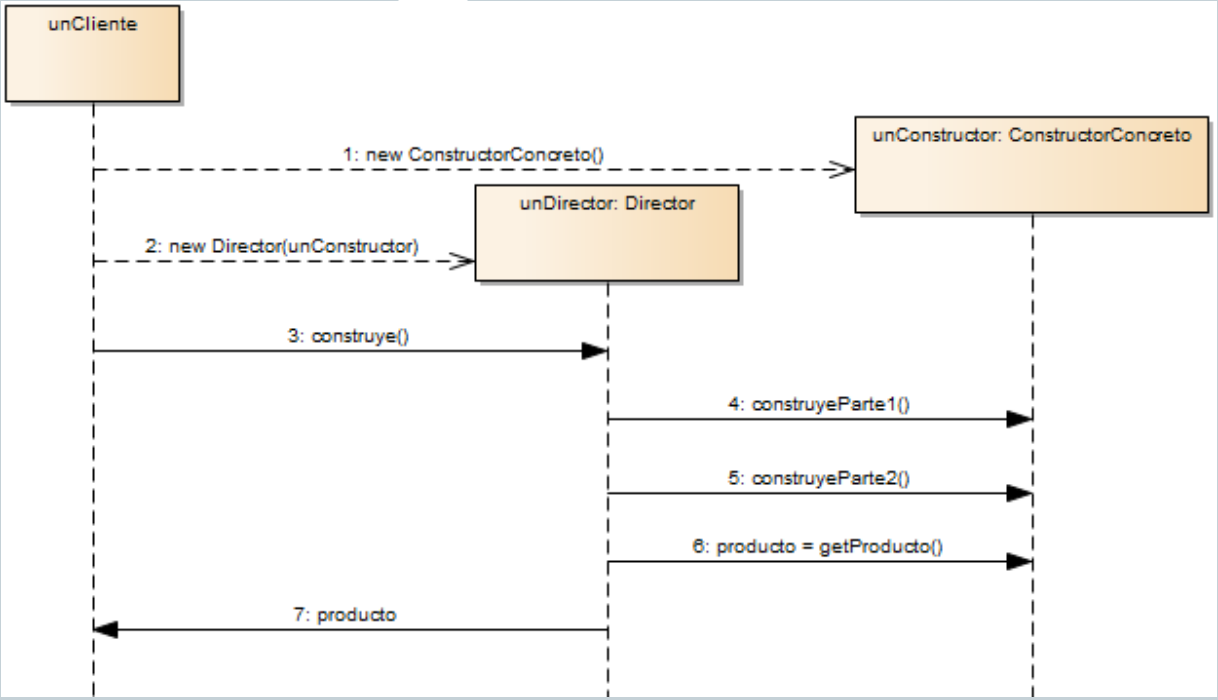
2. Participantes

- Los participantes del patrón son los siguientes:
  - ConstructorAbstracto (ConstructorDocumentacionVehiculo) es la clase que define la firma de los métodos que construyen las distintas partes del producto, así como la firma del método que permite obtener el producto, una vez construido.
  - ConstructorConcreto (ConstructorDocumentacionVehiculoHtml y ConstructorDocumentacionVehiculoPdf) es la clase concreta que implementa los métodos del constructor abstracto.
  - Producto (Documentacion) es la clase que define el producto. Puede ser abstracta y poseer varias subclases concretas (DocumentacionHtml y DocumentacionPdf) en caso de implementaciones diferentes.
  - Director es la clase que se encarga de construir el producto a partir de la interfaz del constructor abstracto.

3. Colaboraciones

- El cliente crea un constructor concreto y un director. El director construye, bajo demanda del cliente, invocando al constructor y reenvía el resultado al cliente.
- La siguiente figura ilustra este funcionamiento con un diagrama de secuencia UML.

El patrón Builder, IV



4. Dominios de uso

- El patrón se utiliza en los siguiente dominios:
  - Un cliente necesita construir objetos complejos sin conocer su implementación.
  - Un cliente necesita construir objetos complejos que tienen varias representaciones o implementaciones

El patrón Builder, V

5. Ejemplo en Java

- Presentamos a continuación un ejemplo de uso del patrón escrito en Java. El código Java correspondiente a la clase abstracta Documentacion y sus subclases aparecen a continuación.
- Por motivos de simplicidad, los documentos son cadenas de caracteres para la documentación en formato HTML y PDF. El método imprime muestra las distintas cadenas de caracteres que representan los documentos.

El patrón Factory Method, I

Descripción

- El objetivo del patrón Factory Method es proveer un método abstracto de creación de un objeto delegando en las subclases concretas su creación efectiva.

Ejemplo

- Vamos a centrarnos en los clientes y sus pedidos. La clase Cliente implementa el método creaPedido que debe crear el pedido. Ciertos clientes solicitan un vehículo pagando al contado y otros clientes utilizan un crédito.
- En función de la naturaleza del cliente, el método creaPedido debe crear una instancia de la clase PedidoContado o una instancia de la clase PedidoCredito. Para realizar estas alternativas, el método creaPedido es abstracto. Ambos tipos de cliente se distinguen mediante dos subclases concretas de la clase abstracta Cliente:
  - La clase concreta ClienteContado cuyo método creaPedido crea una instancia de la clase PedidoContado.
  - La clase concreta ClienteCredito cuyo método creaPedido crea una instancia de la clase PedidoCredito.

El patrón Factory Method, II

- Tal diseño está basado en el patrón Factory Method, el método creaPedido es el método de fabricación. El ejemplo se detalla en la figura del acetato 22.

Estructura

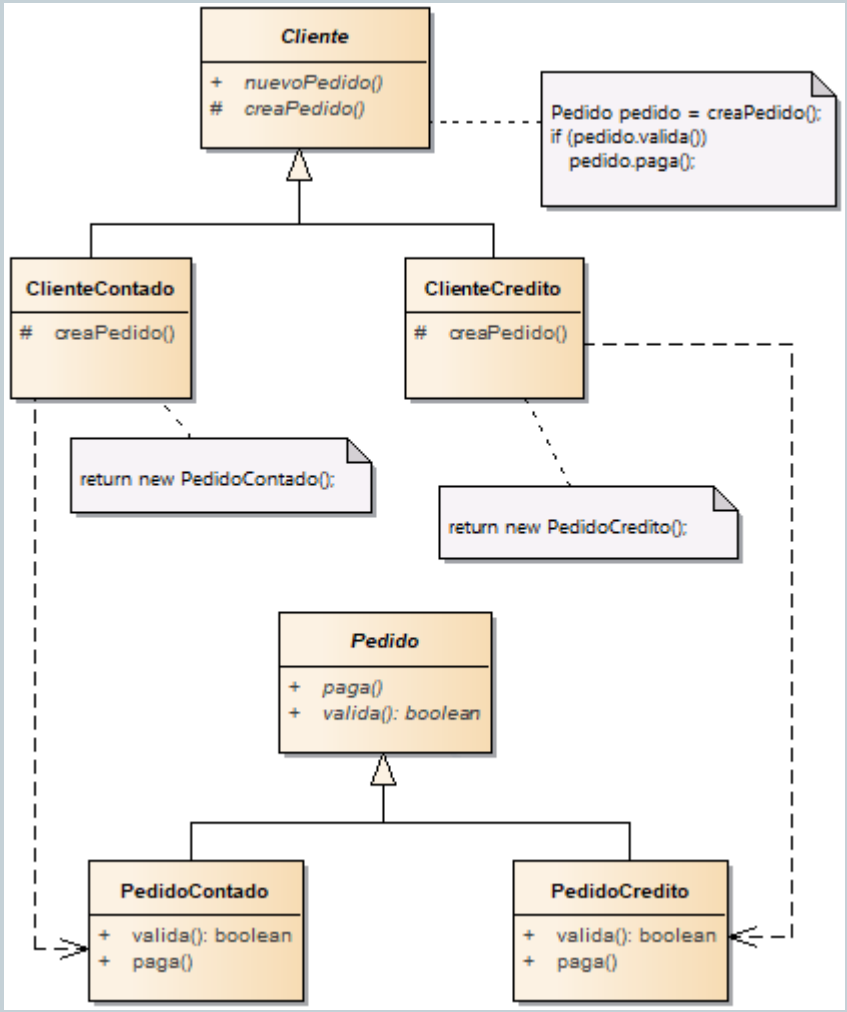
1. Diagrama de clases

- La figura del acetato 23 detalla la estructura genética del patrón.

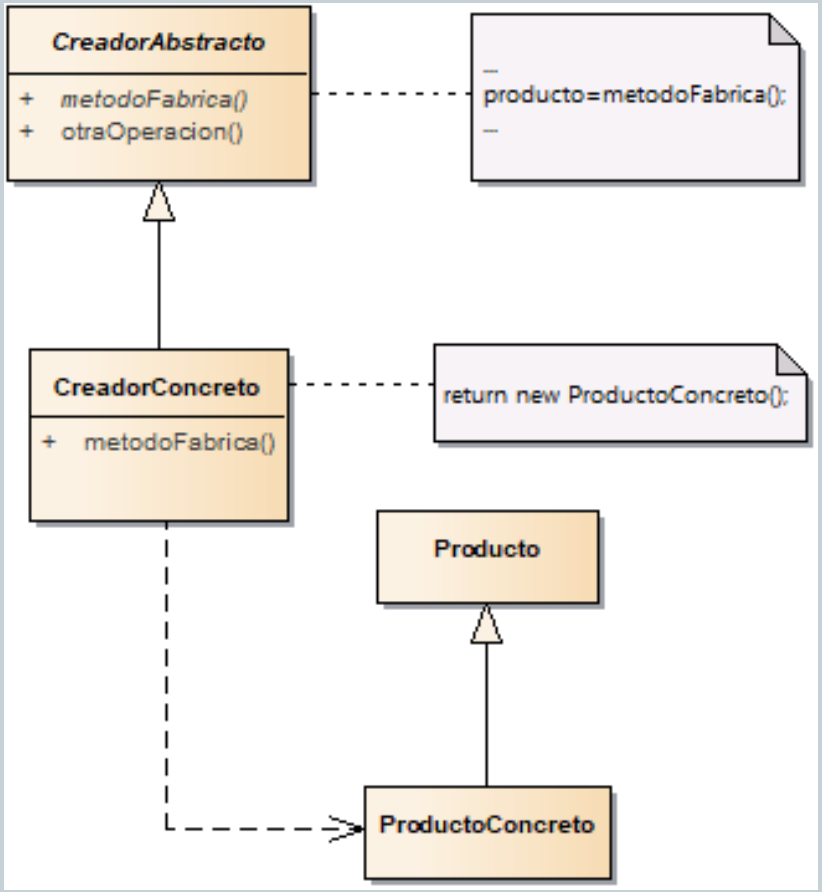
2. Participantes

- Los participantes en el patrón son los siguientes:
  - CreadorAbstracto (Cliente) es una clase abstracta que introduce la firma del método de fabrica y la implementación de métodos que invocan el método fabrica.
  - CreadorConcreto (ClienteContado, ClienteCredito) es una clase concreta que implementa el método fabrica. Pueden existir varios creadores concretos.
  - Producto (Pedido) es una clase abstracta que describe las propiedades comunes de los productos.
  - ProductoConcreto (PedidoContado, PedidoCredito) es una clase concreta que describe completamente un producto.

El patrón Factory Method



Estructura del patrón Factory Method



El patrón Factory Method, III

3. Colaboraciones

- Los métodos concretos de la clase CreadorAbstracto se basan en la implementación del método de fabrica en las sub clases. Esta implementación crea una instancia de la sub clase adecuada de Producto.

4. Dominios de uso

- El patrón se utiliza en los siguientes casos:
  - Una clase que sólo conoce los objetos con los que tiene relaciones.
  - Una clase quiere transmitir a sus subclases las elecciones de instanciación aprovechando un mecanismo de polimorfismo.

5. Ejemplo en Java

- El código fuente de la clase abstracta Pedido y de sus dos subclases concretas aparece a continuación. El importe del pedido se pasa como parámetro al constructor de la clase. Si la validación de un pedido al contado es sistemática, tenemos la posibilidad de escoger, para nuestro ejemplo, aceptar únicamente aquellos pedidos provistos de un crédito cuyo valor se sitúe entre 1,000 y 5,000.

El patrón Prototype, I

Descripción

- El objetivo de este patrón es la creación de nuevos objetos mediante duplicación de objetos existentes llamados prototipos que disponen de la capacidad de clonación.

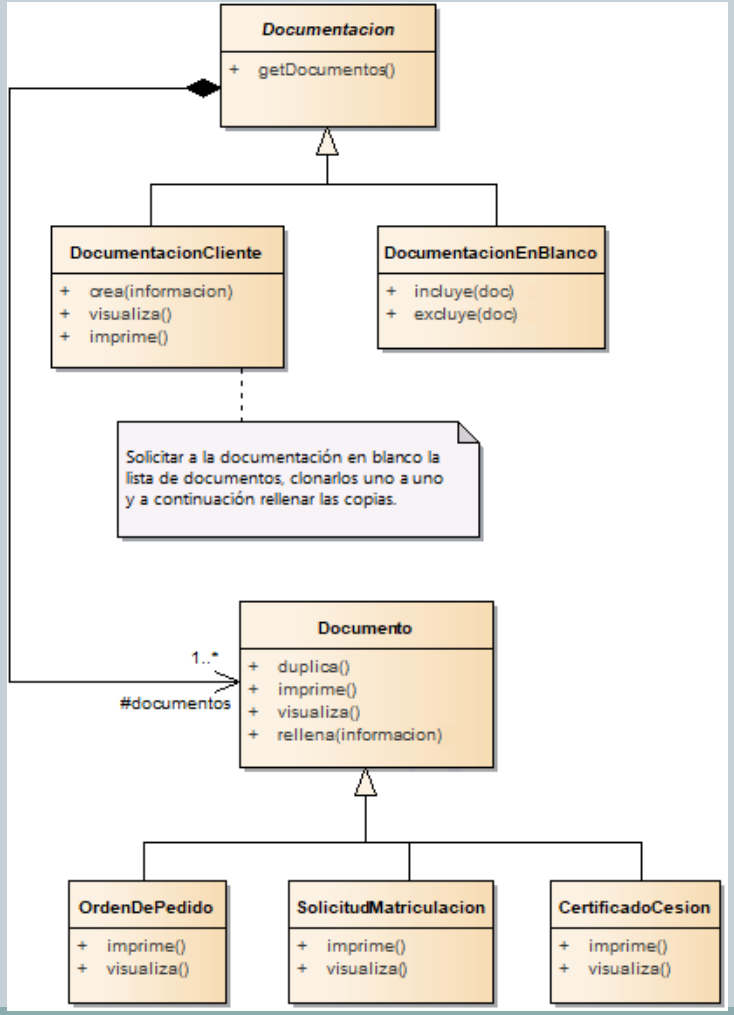
Ejemplo

- Durante la compra de un vehículo, un cliente debe recibir una documentación compuesta por un número concreto de documentos tales como el certificado de cesión, la solicitud de emplacamiento o incluso la orden de pedido.
- Existen otros tipos de documentos que pueden incluirse o excluirse a esta documentación en función de las necesidades de gestión o de cambios de reglamentación.
- Introducimos una clase Documentacion cuyas instancias son documentaciones compuestas por diversos documentos obligatorios. Para cada tipo de documento, incluimos su clase correspondiente.
- A continuación creamos un modelo de documentación que consiste en una instancia particular de la clase Documentacion y que contiene los distintos documentos necesarios, documentos en blanco. Llamamos a esta documentación “documentación en blanco”.

El patrón Prototype, II

- De este modo definimos a nivel de las instancias, y no a nivel de las clases, el contenido preciso de la documentación que debe recibir un cliente. Incluir o excluir un documento en la documentación en blanco no supone ninguna modificación en su clase.
- Una vez presentada la documentación en blanco, recurrimos al proceso de clonación para crear las nuevas documentaciones.
- Cada nueva documentación se crea duplicando todos los documentos de la documentación en blanco.
- Esta técnica basada en objetos que poseen la capacidad de clonación utiliza el patrón Prototype, y los documentos constituyen los distintos prototipos.
- La figura del acetato 27 ilustra este uso. La clase Documento es una clase abstracta conocida por la clase Documentación.
- Sus subclases corresponden a los distintos tipos de documento. Incluyen el método duplica que permite clonar una instancia existente para obtener una nueva.

El patrón Prototype



El patrón Prototype, III

- La clase Documentacion también es abstracta. Posee dos subclases concretas:
  - La clase DocumentacionEnBlanco, que posee una única instancia que contiene todos los documentos necesarios (documentos en blanco). Esta instancia se manipula mediante los métodos incluye y excluye.
  - La clase DocumentacionCliente, cuyo conjunto de documentos se crea solicitando a la única instancia de la clase DocumentacionEnBlanco la lista de documentos en blanco y agregándolos uno a uno tras haberlos clonado.

Estructura

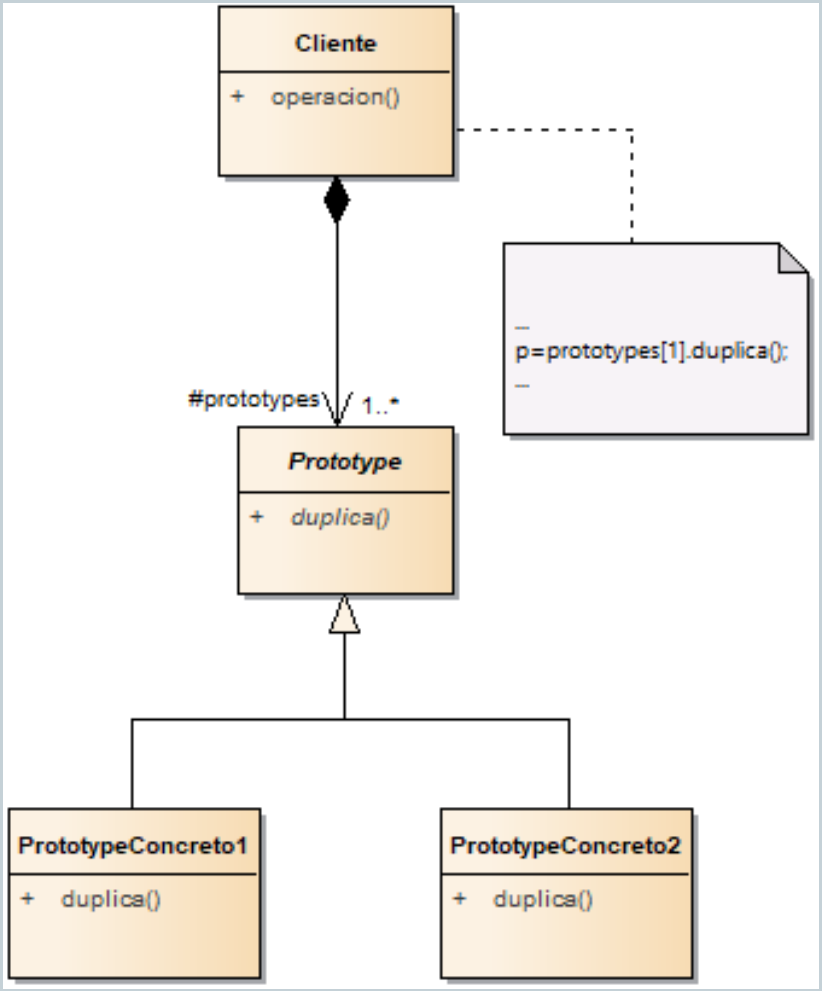
1. Diagrama de clases

- La figura del acetato 29 detalla la estructura genérica del patrón.

2. Participantes

- Los participantes del patrón son los siguientes:
  - Cliente (Documentacion, DocumentacionCliente, DocumentacionEnBlanco) es una clase compuesta por un conjunto de objetos llamados prototipos, instancias de la clase abstracta Prototype. La clase Cliente necesita duplicar estos prototipos sin tener por qué conocer ni la estructura interna del Prototype ni su jerarquía de subclases.

Estructura del patrón Prototype



El patrón Prototype, IV

- Prototype (Documento) es una clase abstracta de objetos capaces de duplicarse a sí mismos. Incluye la firma del método “duplica”.
- PrototypeConcreto1 y PrototypeConcreto2 (OrdenDePedido, SolicitudEmplacamiento, CertificadoCesion) son las subclases concretas de Prototype que definen completamente un prototipo e implementan el método duplica.

3. Colaboración

- El cliente solicita a uno o varios prototipos que se dupliquen a sí mismos.

4. Dominios de uso

- El patrón Prototype se utiliza en los siguientes dominios:
  - Un sistema de objetos debe crear instancias sin conocer la jerarquía de clases que las describe.
  - Un sistema de objetos debe crear instancias de clases dinámicamente.
  - El sistema de objetos debe permanecer simple y no incluir una jerarquía paralela de clases de fabricación.

5. Ejemplo en Java

El patrón Prototype, V

- El código fuente de la clase abstracta Documento y de sus subclases concretas aparece a continuación.
- Para simplificar, a diferencia del diagrama de clases, los métodos duplica y rellena se concretan en la clase Documento. El método duplica utiliza el método clone que proporciona Java.
- El método *clone* de Java nos evita tener que copiar manualmente cada atributo. En consecuencia, es posible implementar el método duplica completamente en la clase abstracta Documento.

El patrón Singleton, I

Descripción

- El patrón Singleton tiene como objetivo asegurar que una clase sólo posee una instancia y proporcionar un método de clase único que devuelva esta instancia.
- En ciertos casos es útil gestionar clases que posean una única instancia. En el marco de los patrones de construcción, podemos citar el caso de una fábrica de productos (patrón Abstract Factory) del que sólo es necesario crear una instancia.

Ejemplo

- En el sistema de venta online de vehículos, debemos gestionar clases que poseen una sola instancia.
- El sistema de documentación que debe entregarse al cliente tras la compra de un vehículo (como el certificado de cesión, la solicitud de emplacamiento y la orden de pedido) utiliza la clase DocumentacionEnBlanco que sólo posee una instancia. Esta instancia referencia todos los documentos necesarios para el cliente. Esta instancia única se llama la documentación en blanco, pues los documentos a los que hace referencia están todos en blanco.

El patrón Singleton, II

- El uso completo de la clase DocumentacionEnBlanco se explica en la parte dedicada al patrón Prototype.
- La figura a) del acetato 34 ilustra el uso del patrón Singleton para la clase DocumentacionEnBlanco. El atributo de clase instance contiene o bien null o bien la única instancia de la clase DocumentacionEnBlanco.
- El método de clase Instance reenvía esta instancia única devolviendo el valor del atributo instance. Si este atributo vale null, se inicializa previamente mediante la creación de la instancia única.

Estructura

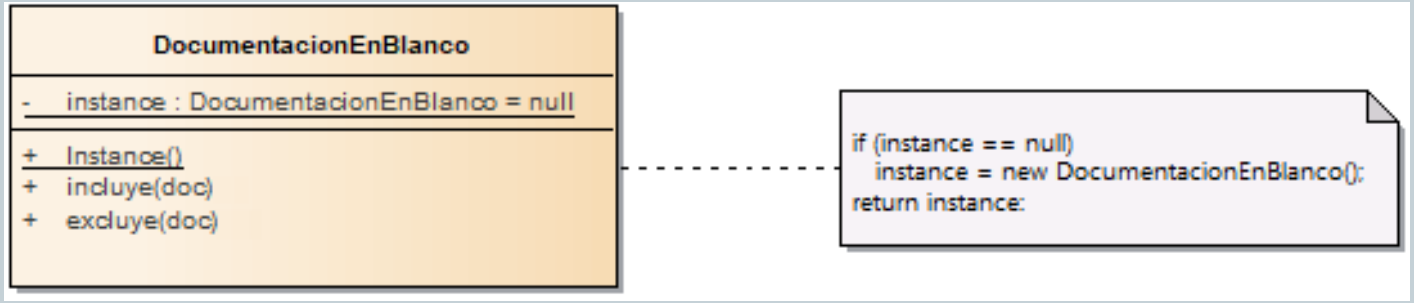
1. Diagrama de clases

- La figura b) del acetato 34 detalla la estructura genérica del patrón-

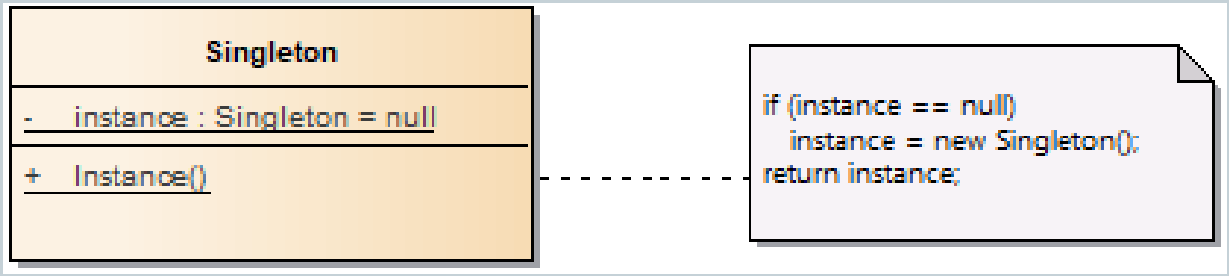
2. Participante

- El único participante es la clase Singleton, que ofrece acceso a la instancia única mediante el método de la clase Instance.

El patrón Singleton y su estructura



a)



b)

El patrón Singleton, III

- Por otro lado, la clase Singleton posee un mecanismo que asegura que sólo puede existir una única instancia. Este mecanismo bloquea la creación de otras instancias.

3. Colaboración

- Cada cliente de la clase Singleton accede a la instancia única mediante el método de clase Instance. No puede crear nuevas instancias utilizando el operador habitual de instanciación (operador new), que está bloqueado.

4. Dominio de uso

- El patrón se utiliza en el siguiente caso:
  - Sólo debe existir una única instancia de una clase.
  - Esta instancia sólo debe estar accesible mediante un método de clase.
- Nota: El uso del patrón Singleton ofrece a su vez la posibilidad de dejar de utilizar variables globales.

5. Ejemplos en Java

5.1 Documentación en blanco

El patrón Singleton, IV

- El código Java completo de la clase DocumentacionEnBlanco aparece en el apartado dedicado al patrón Prototype. La sección de esta clase relativa al uso del patrón Singleton se muestra a continuación.
- El constructor de esta clase tiene una visibilidad privada de modo que sólo pueda utilizarlo el método Instance. De este modo, ningún objeto externo a la clase DocumentacionEnBlanco puede crear una instancia utilizando el operador new.
- Del mismo modo, el atributo _instance también tiene visibilidad privada para que sólo sea posible acceder a él desde el método de la clase Instance.
- Veamos con detalle el código fuente.
- Archivo, DocumentoenBlanco.java

El patrón Singleton, V

- El único cliente de la clase DocumentacionEnBlanco es la clase DocumentacionCliente que, en su constructor, obtiene una referencia a la documentación en blanco invocando al método Instance.
- A continuación, el constructor accede a la lista de documentos en blanco.

```
DocumentacionEnBlanco documentoEnBlanco =
Documentacion.Instance();
List<Documento> documentosEnBlanco =
    documentacionEnBlanco.getDocumentos();
```

5.2 La clase Comercial

- En el sistema de venta de vehículos, queremos representar al vendedor mediante una clase que permita memorizar su información en lugar de utilizar variables globales que contienen respectivamente su nombre, dirección, etc.
- La clase Comercial se describe a continuación (checar el archivo Comercial.java).
- Su ejecución muestra que sólo existe una instancia debido a que el método visualiza de TestComercial no recibe ningún parámetro.