Patrones GOF creacionales

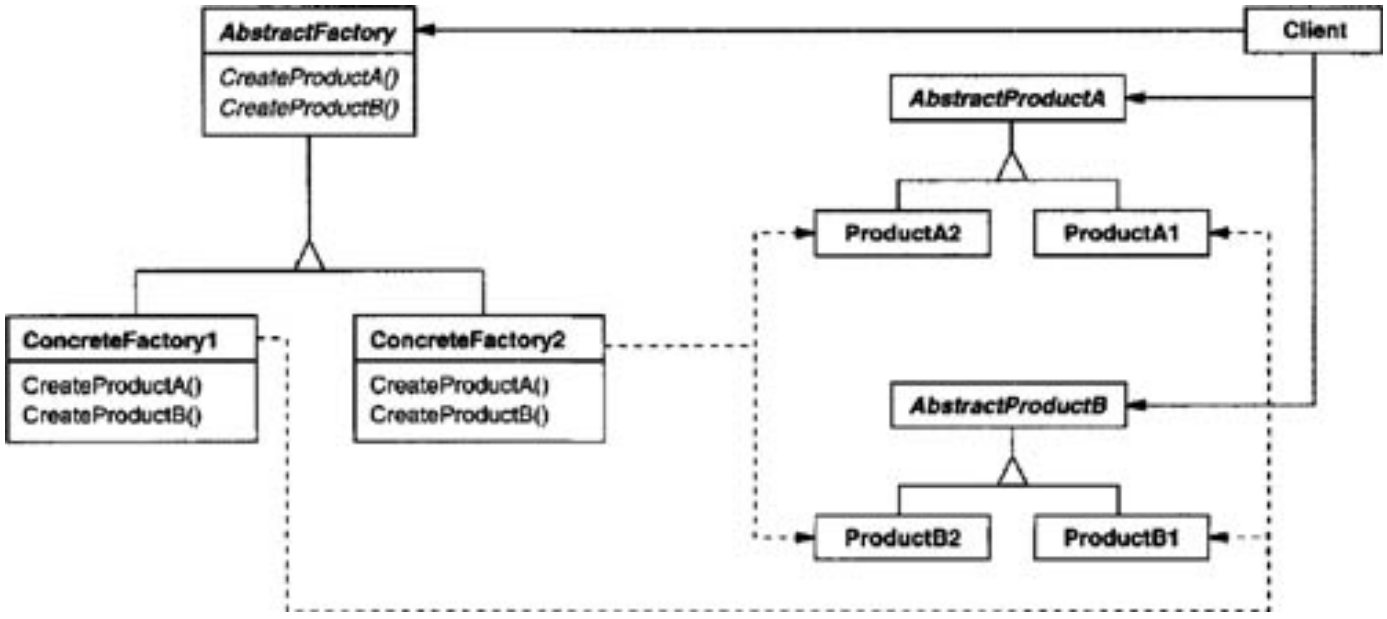
Diseño de Sistemas Software

Patrones GOF creacionales

- Los patrones de diseño creacionales abstraen el proceso de instanciación. Ayudan a independizar el sistema del modo en que los objetos son creados, compuestos y representados.
  - Un **patrón creacional de clases** usa herencia para variar la clase que se instancia
  - Un **patrón creacional de objetos** delega la instanciación a otro objeto

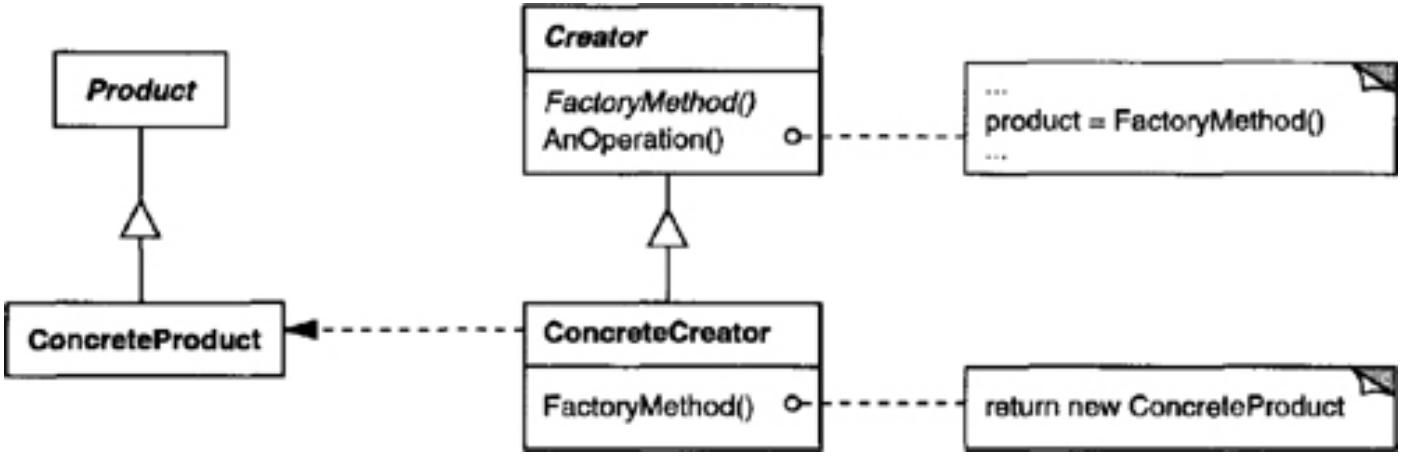
Abstract Factory

- **Abstract Factory**: provee una interfaz para crear familias de objetos producto relacionados o que dependen entre sí, sin especificar sus clases concretas.



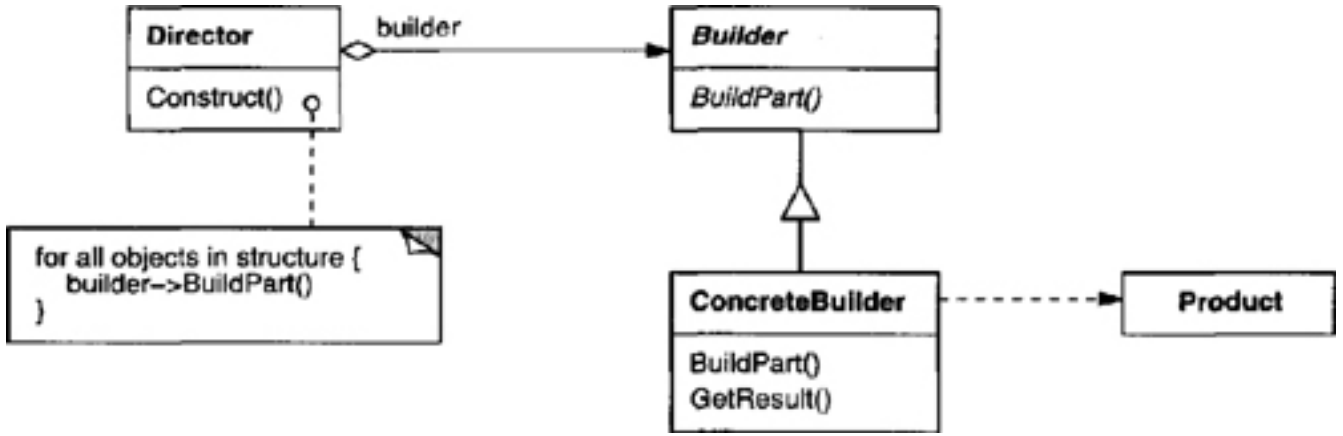
Factory Method

- **Factory Method**: define una interfaz para crear un objeto delegando la decisión de qué clase crear en las subclases. Este enfoque también puede ser llamado constructor “virtual”



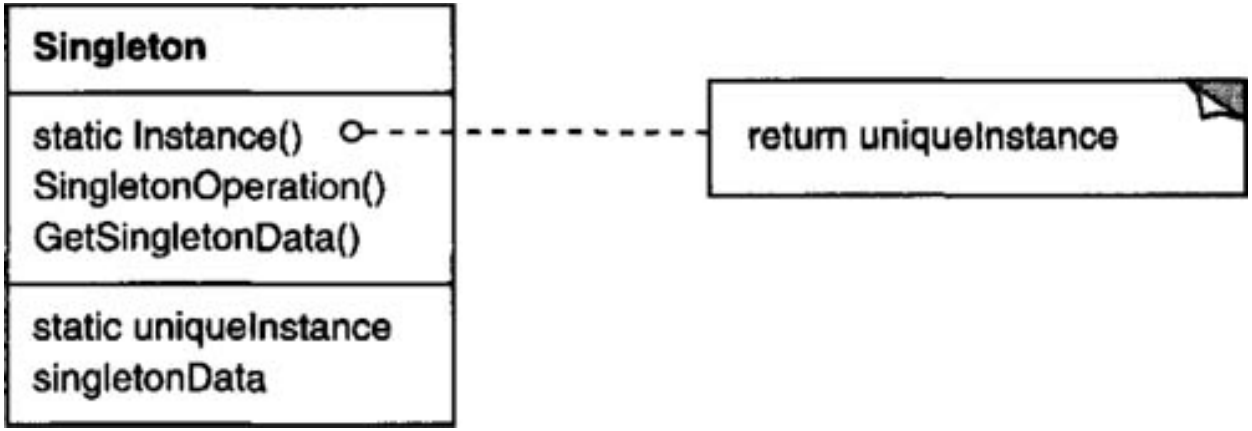
Builder

- **Builder**: separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones. Simplifica la construcción de objetos con estructura interna compleja y permite la construcción de objetos paso a paso.



Singleton

- **Singleton**: asegura que una determinada clase sea instanciada una y sólo una vez, proporcionando un único punto de acceso global a ella.



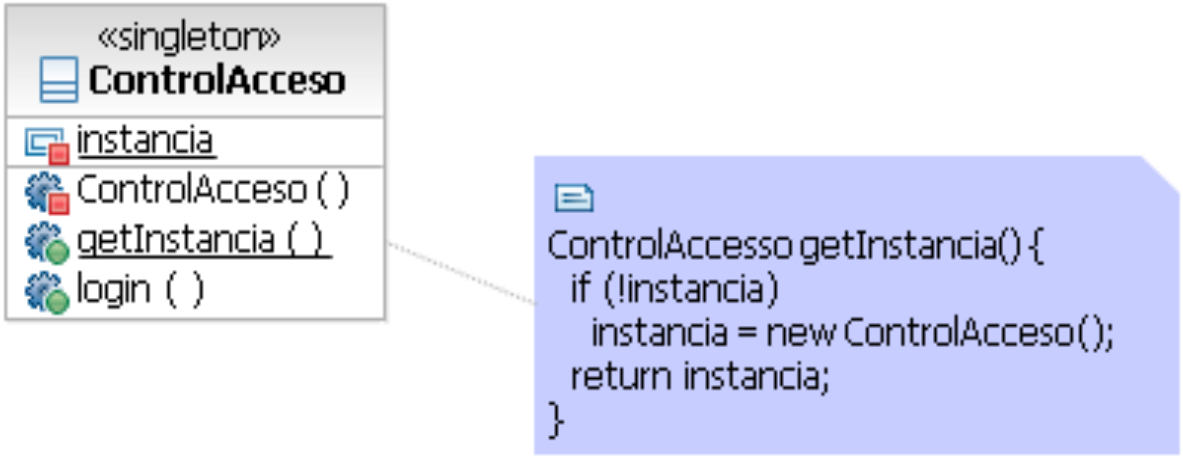
Ejercicios

Ejercicio 1

- **Problema**: nos han pedido introducir en la aplicación que estamos diseñando una clase de login que proporcione un punto de acceso global a la operación con un nombre de usuario y contraseña para todos los componentes de la aplicación. Propón una solución utilizando un patrón GOF.

Ejercicio 1

- **Solución**: patrón Singleton

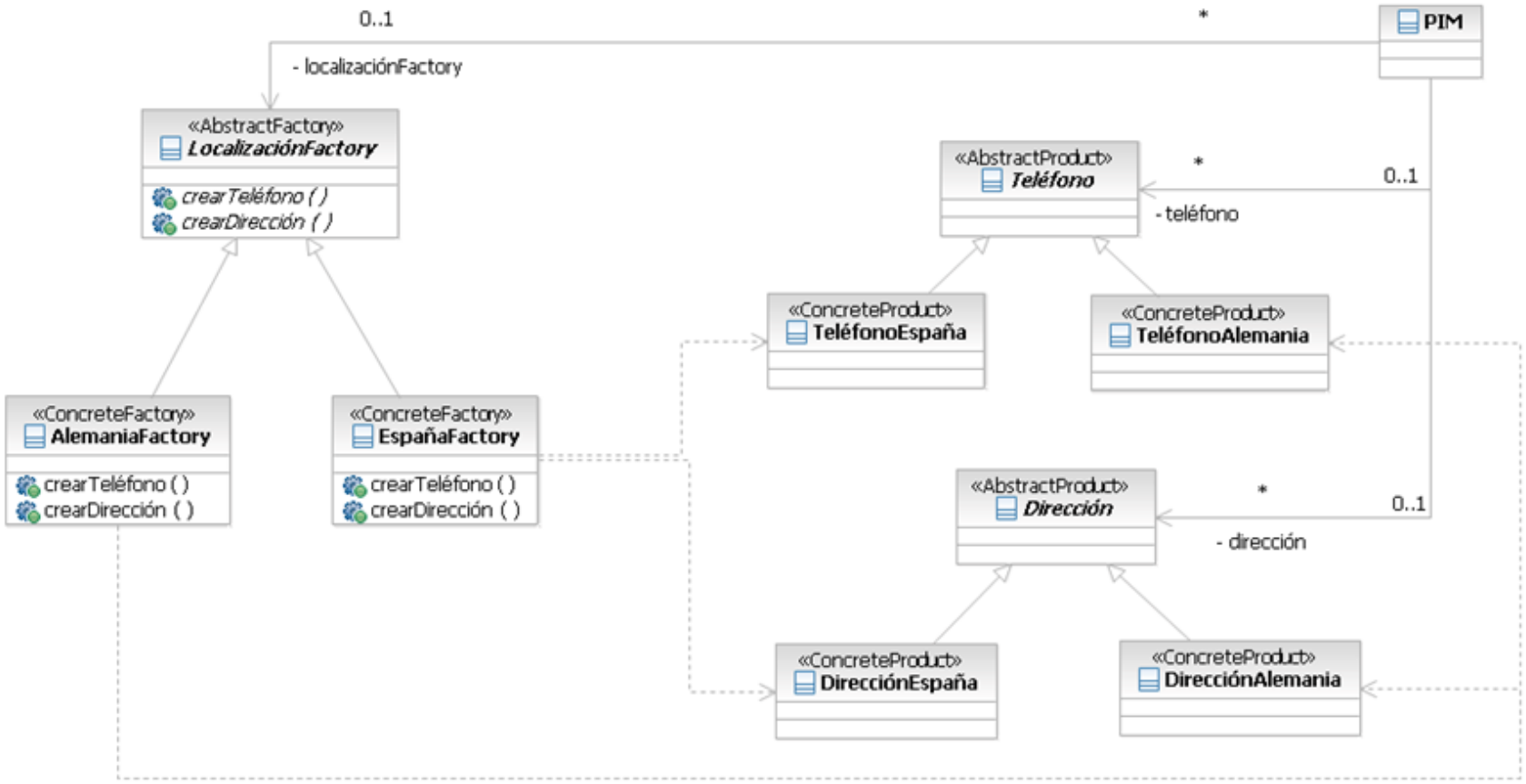


Ejercicio 2

- **Problema**: queremos implementar un manejador de información personal, que controla, entre otras cosas, números de teléfono y direcciones. El alta de números de teléfono sigue una regla particular, que depende de la región y país a la que pertenezca el número. Sabemos que el número de países manejados por la aplicación va a crecer en un futuro, y queremos proteger nuestro código de esa variación. Propón un diseño que, utilizando un patrón GOF, solucione este problema.

Ejercicio 2

- **Solución**: patrón Abstract Factory

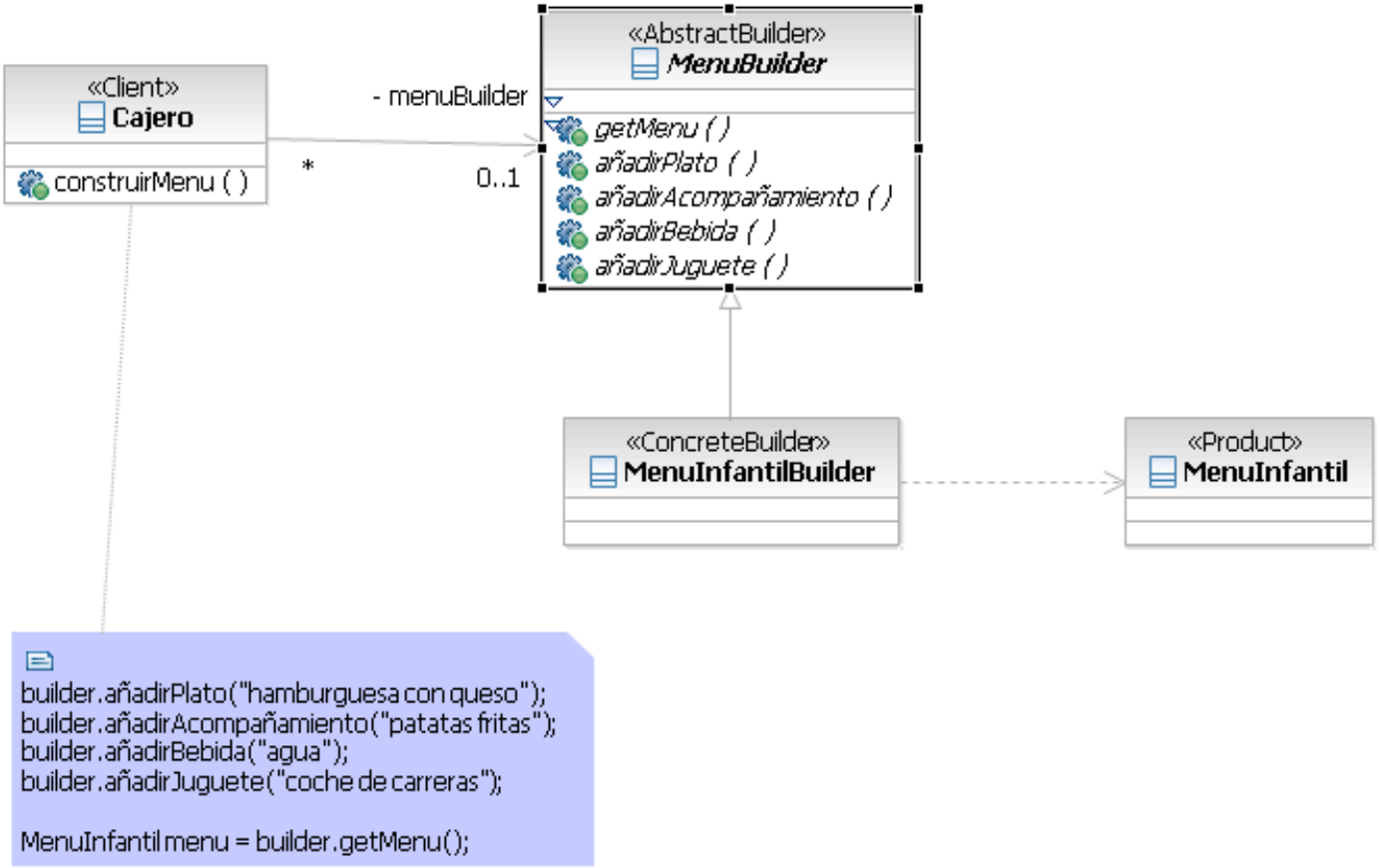


Ejercicio 3

- **Problema**: Imaginad que estamos construyendo una aplicación para restaurantes de comida rápida que preparan, entre otros, menús infantiles. Para ello hemos creado una clase Menú, de la que hereda Menú infantil. Generalmente estos menús están formados por un plato principal, un acompañamiento, una bebida y un juguete. Si bien el contenido del menú puede variar, el proceso de construcción es siempre el mismo: el cajero indica a los empleados los pasos a seguir. Estos pasos son: preparar un plato principal, preparar un acompañamiento, incluir un juguete y guardarlos en una bolsa. La bebida se sirve en un vaso y queda fuera de la bolsa. Completad el diseño.

Ejercicio 3

- **Solución**: patrón Builder

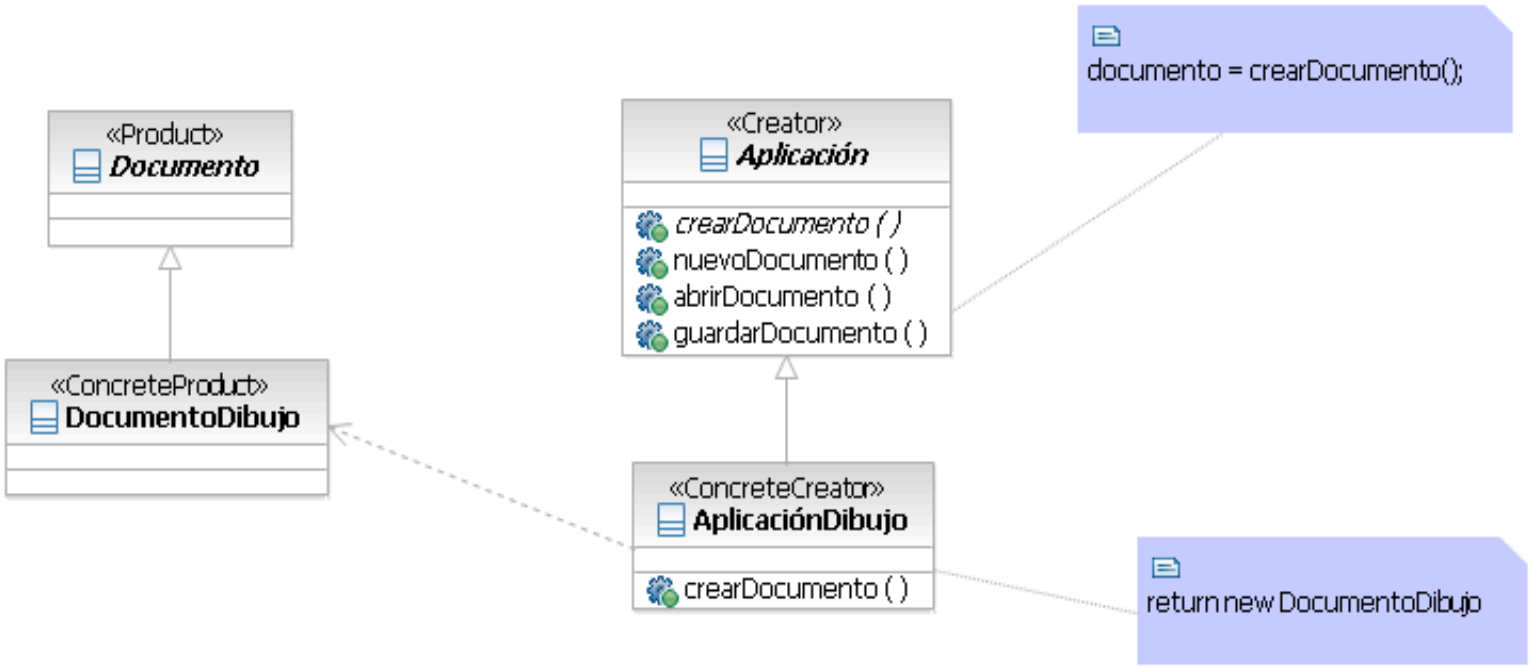


Ejercicio 4

- **Problema**: Nos han pedido colaborar en la elaboración de un framework para gestionar aplicaciones de manejo de documentos. Este framework debe asegurar, entre otras cosas, el correcto manejo de las peticiones de apertura, creación y salvado de documentos de cualquier tipo, donde el tipo concreto dependerá de la aplicación que extienda nuestro framework. Por ejemplo, una aplicación de dibujo contendrá una clase AplicaciónDibujo, y una clase DocumentoDibujo.

Ejercicio 4

- **Solución**: patrón Factory Method



Bibliography

- *Design Patterns: Elements of Reusable Object-Oriented Software*. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley Professional, 1994.