**Tema: Patrones de diseño**

[Factory Method](https://migranitodejava.blogspot.com/2011/05/factory-method.html)

Libera al desarrollador sobre la forma correcta de crear objetos. Define la interfaz de creación de un cierto tipo de objeto, permitiendo que las subclases decidan que clase concreta necesitan instancias.  
Muchas veces ocurre que una clase no puede anticipar el tipo de objetos que debe crear, ya que la jerarquía de clases que tiene requiere que deba delegar la responsabilidad a una subclase.  
  
Este patrón debe ser utilizado cuando:

* Una clase no puede anticipar el tipo de objeto que debe crear y quiere que sus subclases especifiquen dichos objetos.
* Hay clases que delegan responsabilidades en una o varias subclases. Una aplicación es grande y compleja y posee muchos patrones creacionales.

 Diagrama  UML.

A diagram of a product

Description automatically generated

Creator: declara el método de fabricación (creación), que devuelve un objeto de tipo Product.  
ConcretCreator: redefine el método de fabricación para devolver un producto.  
ProductoConcreto: es el resultado final. El creador se apoya en sus subclases para definir el método de fabricación que devuelve el objeto apropiado.  
  
  
Ejemplo  
  
En nuestro ejemplo tenemos una clase abstracta llamada Triangulo, de la cual heredan los 3 tipos de triangulos conocidos.

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

Pero tenemos el siguiente inconveniente: quien se encargue de crear un tipo de triángulo concreto no debería tener que conocer cómo se compone internamente.  
Para ello, hemos creado la clase TrianguloFactory con su correspondiente interface.

A close up of a text

Description automatically generated

A computer code with black text

Description automatically generated

Veremos que, desde el punto de vista del cliente, es muy sencillo poder crear un triángulo:

A screenshot of a computer program

Description automatically generated

Consecuencias

* Como ventaja se destaca que elimina la necesidad de introducir clases específicas en el código del creador. Solo maneja la interfaz Product, por lo que permite añadir cualquier clase ConcretProduct definida por el usuario.
* Por otro lado, es más flexible crear un objeto con un Factory Method que directamente: un método factoría puede dar una implementación por defecto.

Temas a tener en cuenta.  
  
Si bien he escrito sobre la teoría del patrón Factory, hay que decir que en la realidad este patrón es mucho más versátil. Con esto me refiero a que no siempre se trata de casos como el que vimos en el ejemplo (caso ideal), donde el factory se aplica a una jerarquía de subclases.  
Muchas veces nos vamos a encontrar con casos donde no aplican ninguno de los patrones creacionales, pero aún así nos vamos a dar cuenta que necesitamos aliviarle la tarea a un cliente para que pueda crear un determinado objeto. A esta clase que le facilita la creación al cliente también se la suele denominar Factory, aún cuando no se trate de una jerarquía. Como consecuencia, podriamos redefinir el término del factory ajustándolo un poco a la realidad de la programación: se trata de aquella clase que intenta aliviar la carga de crear un objeto de manera correcta, cuando el resto de los patrones creacionales no aplican.