Ejemplo: Escribir en una consola

Imaginá que querés escribir en consola, y para ello deseás crear una clase a parte. Si usáramos la técnica de Factory Method, nos quedaría una jerarquía tal que:

1. Cliente que usa el Factory (ClientLogger.java).
2. El Logger Factory (LoggerCreator.java).
3. Instancia un nuevo logger (Logger.java).
4. Crea un nuevo logger del tipo Logger (ConcreteLogger).

¿Por qué NO entra el patrón Factory Method?

- En este ejemplo, no tenemos distintos tipos de Logger. Se asume que no tenemos distintas variantes, (como podrían ser FilerLogger, RemoteLogger, etc) por lo que no sería necesario. Es decir, la creación del objeto no varía.
- La creación no tiene ninguna condición en especial para crear la clase Logger. No hay ninguna lógica compleja que se cree o no.
- En nuestro enunciado, no parece haber intención de que algún futuro se extienda el código. No habría múltiples implementaciones de Logger, así que crear un clase aparte sería sobre ingeniería.
- Se complejiza el código sin motivo real (como vimos anteriormente). Esto también influye en que no sea flexible el código, que es una parte crucial del mismo.

Una forma más eficiente de realizar este ejemplo es simplemente una sola clase, Logger, tal que cree el nuevo objeto con el que se escribirá en la consola (LoggerFlex.java)

En general, NO usa Factory Method cuando:

1. Si solo necesitas instanciar una clase directamente (hacer new Clase()), el Factory Method agrega complejidad innecesaria.
2. Si tu código no está sujeto a cambios ni extensiones (como nuevos tipos de objetos), una fábrica anticipa una flexibilidad que no se necesita.
3. Si no hay múltiples subtipos posibles que se instancien de manera dinámica, el Factory Method no tiene sentido.
4. El Factory Method se basa en herencia. Si tu diseño favorece la usar objetos dentro de otros, no extenderlos, puede que un patrón como Strategy o Builder sea más adecuado.
5. El Factory Method está ligado a clases base y subclases. Si quieres una lógica de creación completamente separada, el patrón *Factory Object* (o *Factory Class*) o incluso un *Builder* puede ser mejor.