Diseño y Arquitectura de Software

Patrón de Diseño:

"*Abstract Factory*"

David Israel Pérez Montes

19/09/18

Nombre y Clasificación:

“Abstract Factory”

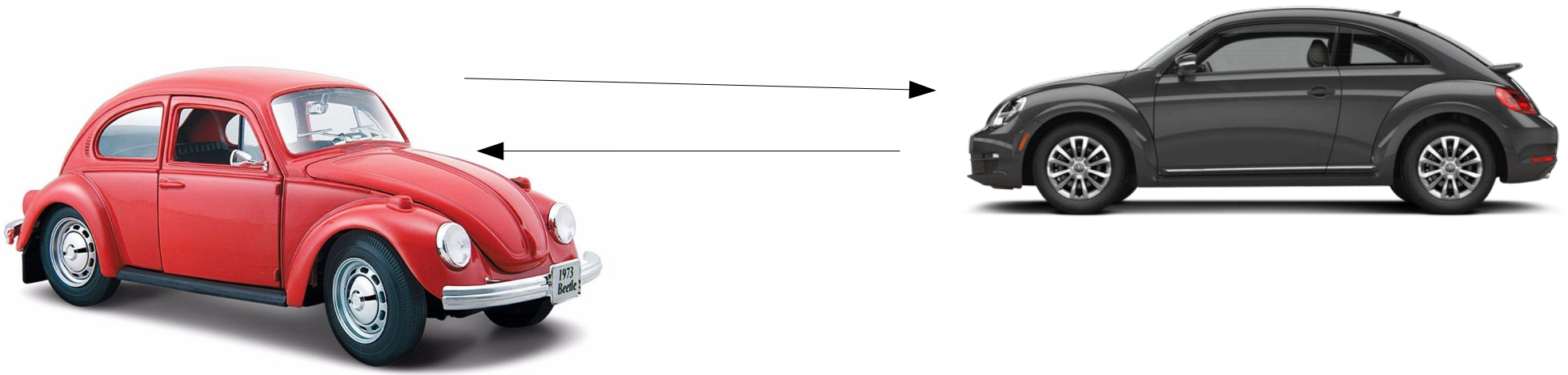
- Se clasifican dentro de los *“Patrones Creacionales”*.
- Estos patrones normalmente trabajan con interfaces, por lo que la implementación concreta que utilicemos queda *aislada.*

Los Patrones Creacionales:

- Facilitan la tarea de crear nuevos objetos, *encapsulando* el proceso.
- *Ocultar* cómo las implementaciones concretas necesitan ser creadas y cómo se combinan entre sí.

Intención

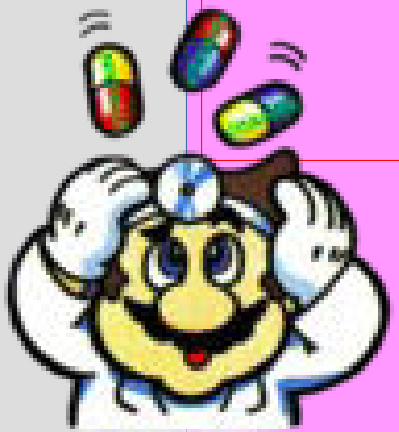
- Abstract Factory nos provee de *una interfaz* que permite la creación de un conjunto de objetos relacionados sin necesidad de especificar en ningún momento cuáles son las implementaciones concretas.



Otros Nombres:

“Abstract Factory”

- También se le conoce como KIT Pattern.



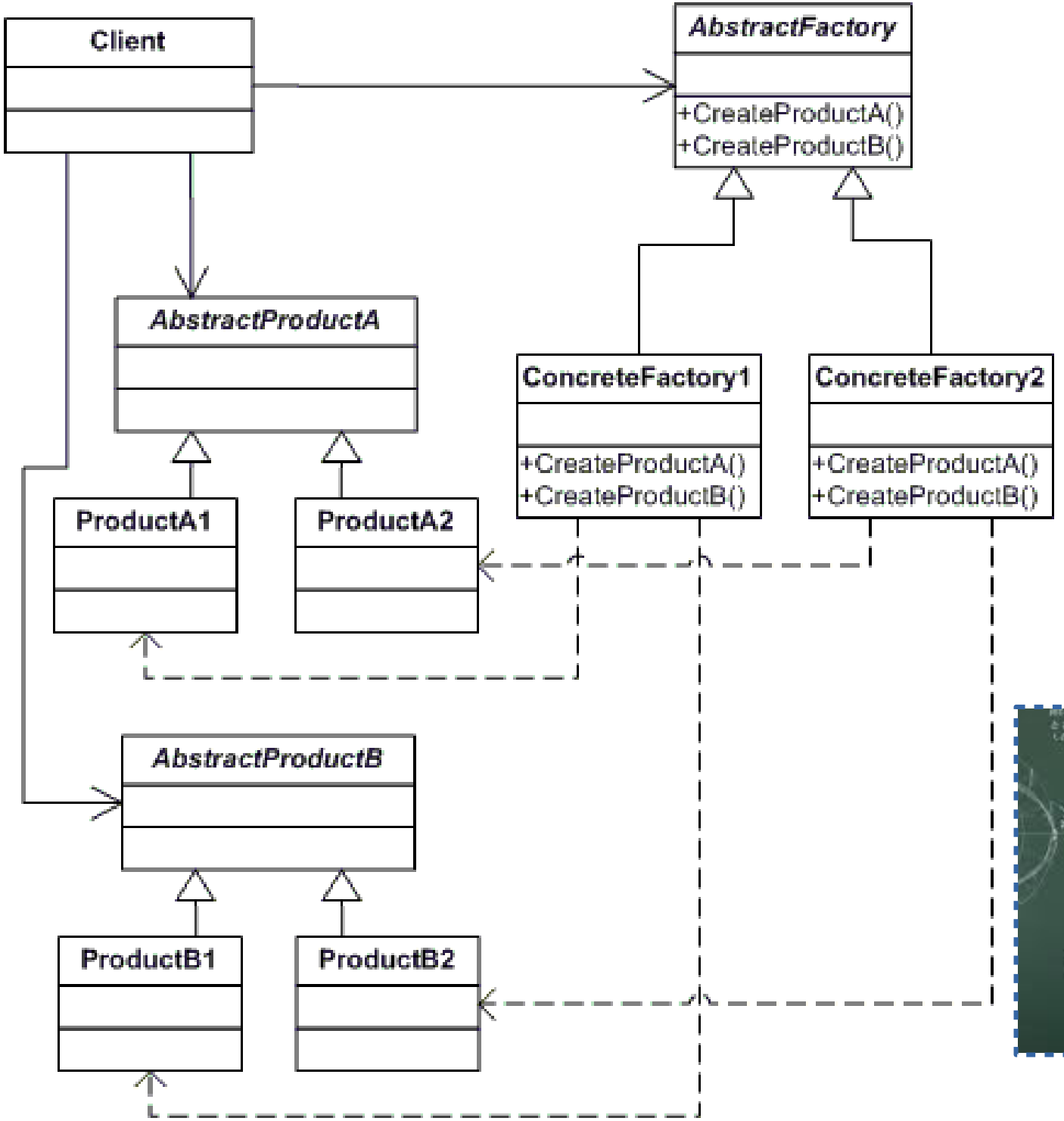
Motivación

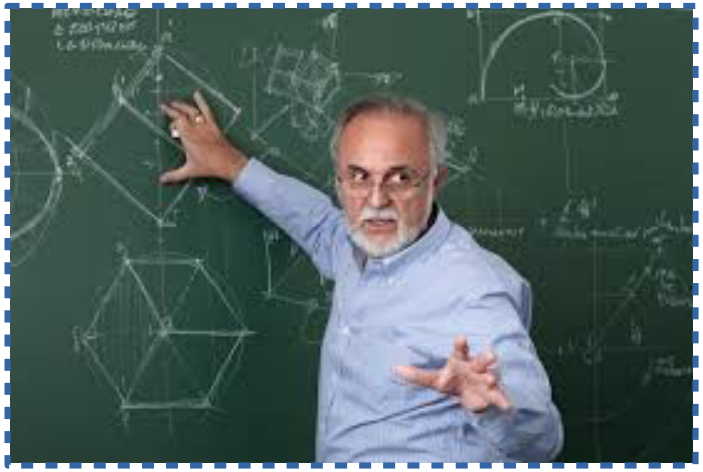
- Debemos crear diferentes objetos, todos pertenecientes a la misma familia.
- El problema que intenta solucionar este patrón es el de crear diferentes familias de objetos.
- El patrón Abstract Factory está aconsejado cuando se prevé la inclusión de nuevas familias de productos, pero puede resultar contraproducente cuando se añaden nuevos productos o cambian los existentes, puesto que afectaría a todas las familias creadas.

Aplicación

- Un sistema que debe ser independiente de la forma en que sus productos son creados, compuestos, y representados.
- Cualquier código que necesite ampliar escala es buen candidato para este patrón.

Estructura UML



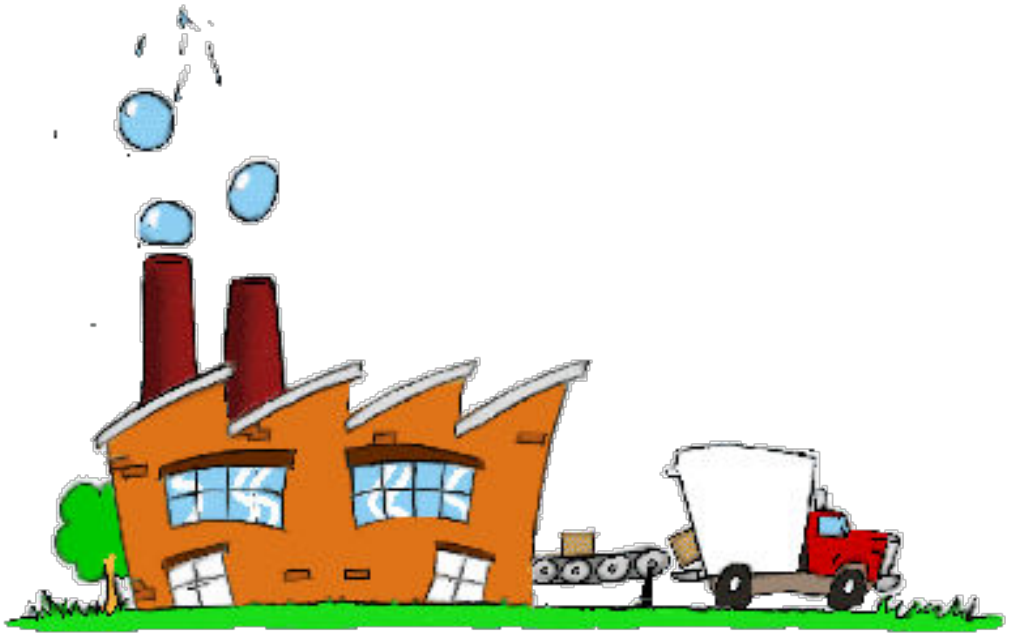


Participantes:

- ***Cliente:*** La clase que llamará a la factoría adecuada ya que necesita crear uno de los objetos que provee la factoría.
- ***Abstract Factory:*** Es la definición de la interfaces de las factorías.
- ***Factorías Concretas:*** Estas son las diferentes familias de productos. Provee de la instancia concreta de la que se encarga de crear.
- ***Producto abstracto:*** Definición de las interfaces para la familia de productos. El cliente trabajará directamente sobre esta interfaz, que será implementada por los diferentes productos concretos.
- ***Producto concreto:*** Implementación de los diferentes productos. Trabajará directamente sobre la super - clase o interfaz.

Colaboraciones:

- Normalmente se crea una única Fábrica Concreta, que se encarga de crear los productos concretos.
- La Fábrica Abstracta difiere la creación de productos a sus subclases.



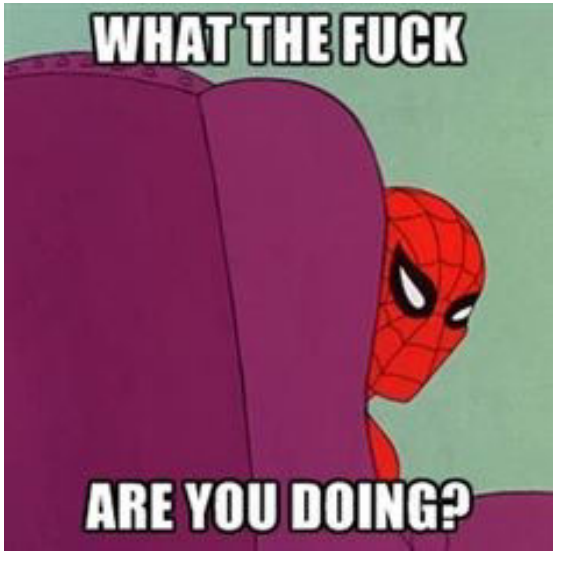
Consecuencias:

Pros:

- Facilita el intercambio de familias de productos.
- Simplifica consistencia entre producto.

Contras:

- Aísla clases concretas del cliente.
- Difícil añadir nuevas clases de productos.



Implementación:

- Si no se conoce bien la estructura del patrón o específicamente a qué se va a aplicar, es muy posible que su implementación no sea del todo optima.
- Por eso hay que estar consiente de cómo se va a aplicar Abstract Factory en el sistema.

Ahora un ejemplo en Python :D

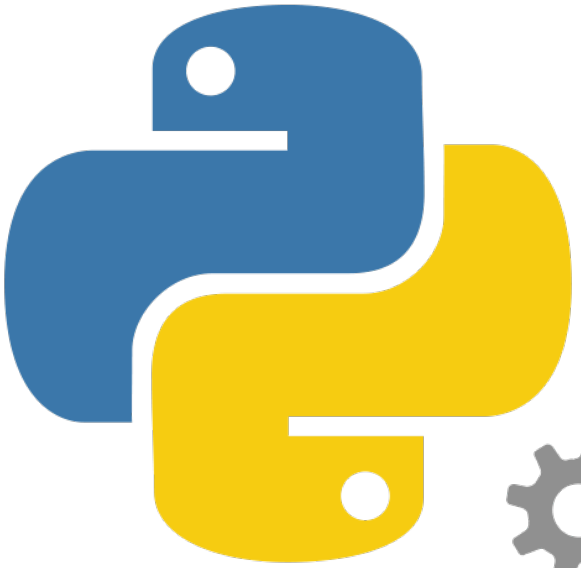
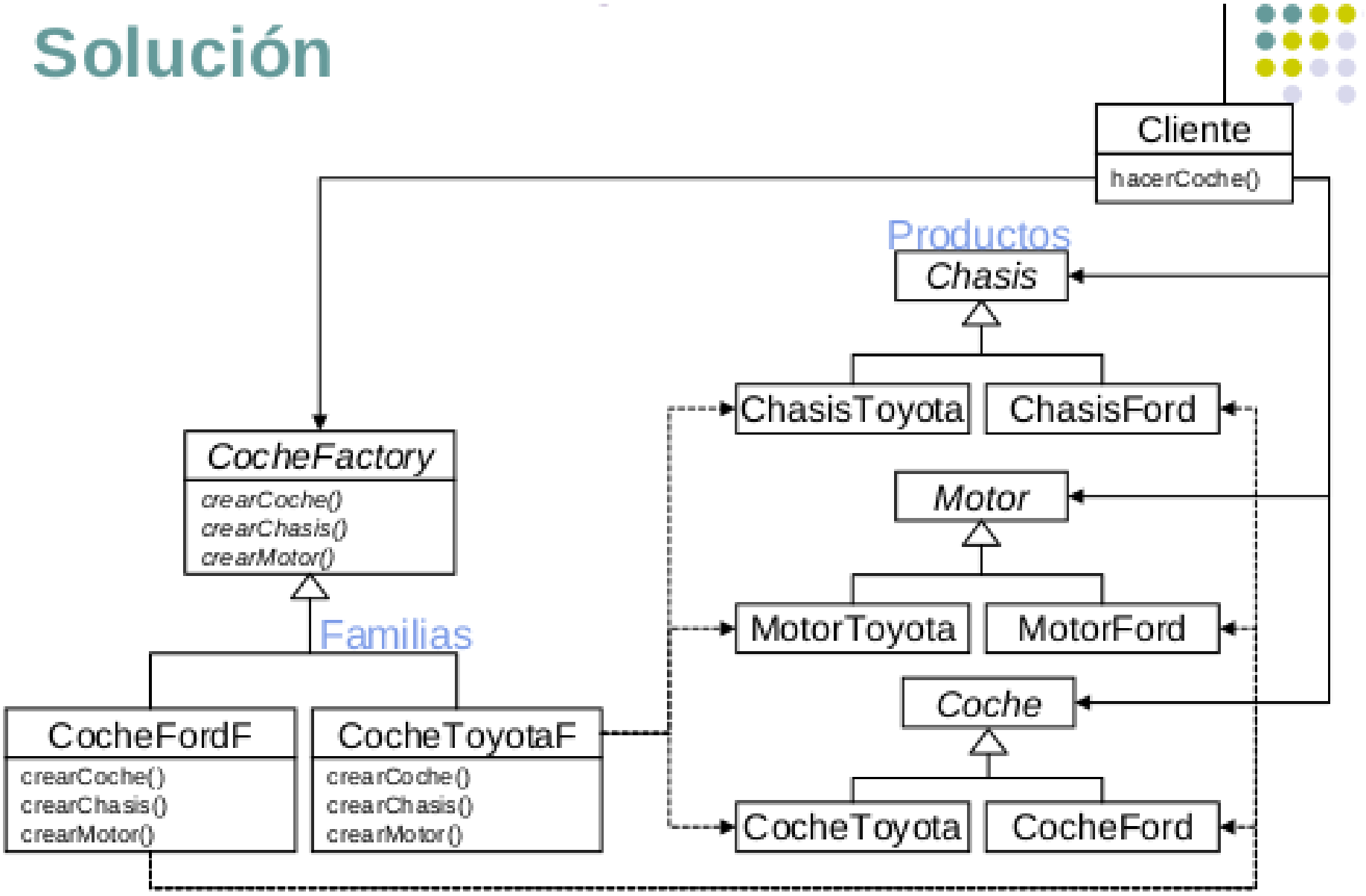


Diagrama para el Ejemplo:

Solución



Usos Conocidos:

- ET++ usa Abstract Factory para archivar portablemente sobre diferentes sistemas Windows (X Windows y SunView)
- La clase base abstracta de Windows System define la interfaz para crear objetos que representan recursos del sistema Windows.

Patrones Relacionados

- Se pueden implementar con Factory Method o Prototype.
- Las factorías concretas suelen ser Singleton.

Fuentes:

- https://www.youtube.com/watch?v=cs9a25KvScM
- https://devexperto.com/patrones-de-diseno-software/
- “Patrones de Diseño: Patrones de creación” Técnicas de Programación - Curso 2007/08
- “Your Brain on Design Patterns: Head first design patterns”, Eric Freeman & Elizabeth Freeman,O’Reilly, 2004.