INSTITUTO TECNOLÓGICO DE TIJUANA
SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN
SEMESTRE AGOSTO-DICIEMBRE 2017.

CARRERA: Ing. Sistemas Computacionales.
MATERIA: PATRONES DE DISEÑO.
SERIE: **ADF-1701**

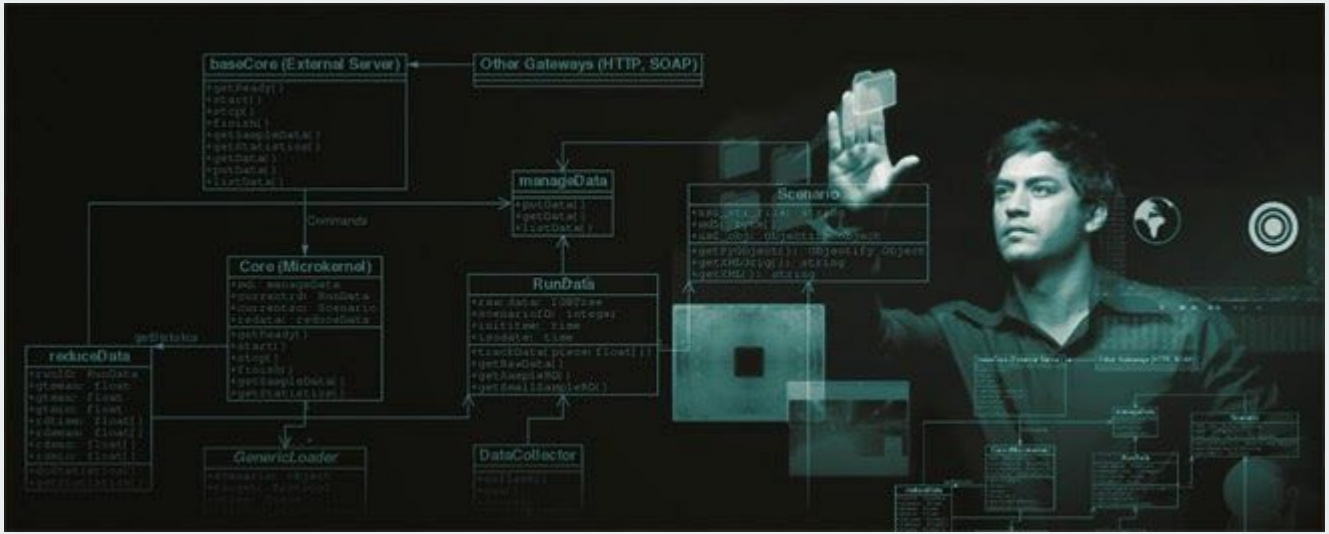
CATEDRÁTICO: Gabriela Lourdes Tapia González.

NOMBRE DEL ALUMNO:

Álvarez Ortega Estephania **No.Control.** 14210446

González Esparza Luis Manuel **No.Control** 13211492

PATRÓN ABSTRACT FACTORY



CUESTIONARIO

1. ¿Cuál es el propósito del Patrón Abstract Factory?
2. ¿En qué se podría aplicar el Patrón Abstract Factory?
3. Menciona las clases que componen el Patrón Abstract Factory.
4. ¿Cuáles son las ventajas del Patrón Abstract Factory?
5. ¿Cuál es la desventaja del Patrón Abstract Factory?

PROPÓSITO

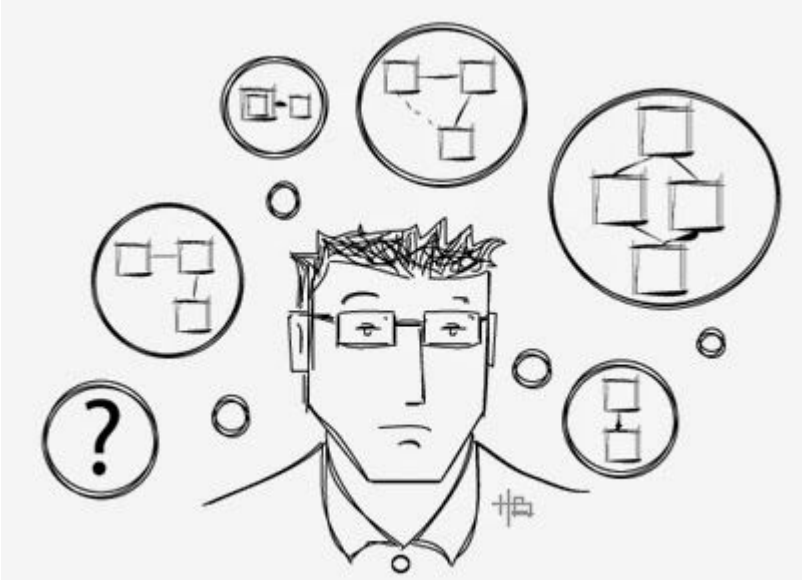
Proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

APLICABILIDAD

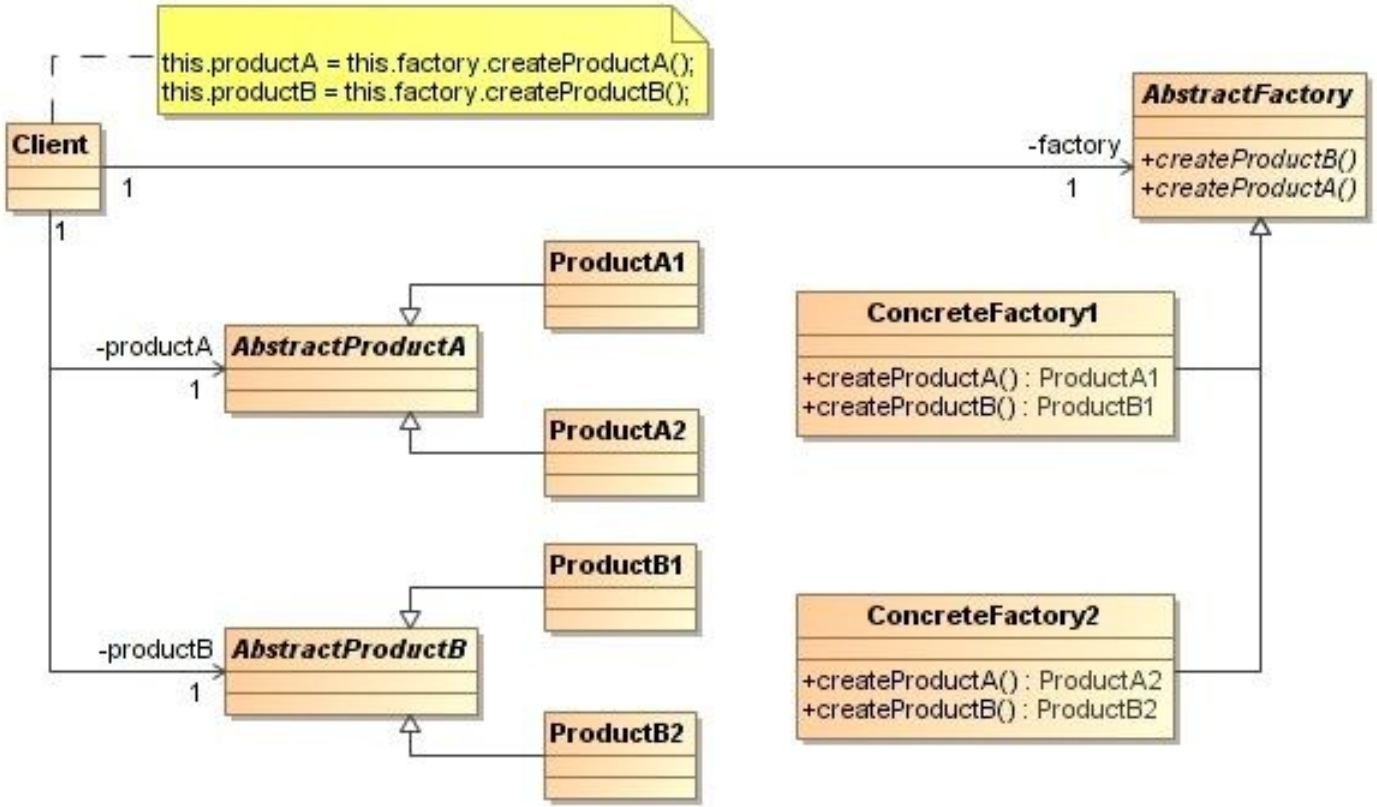
- ➔ Un sistema debe ser independiente de cómo se crean, componen y representan sus productos.
- ➔ Un sistema debe ser configurado con una familia de productos de entre varias.
- ➔ Una familia de objetos producto relacionados está diseñada para ser usada conjuntamente, y es necesario hacer cumplir esta restricción.
- ➔ Quiere proporcionar una biblioteca de clases de productos, y sólo quiere relevar sus interfaces, no sus implementaciones.

CLASES

- AbstractFactory.
- ConcreteFactory
- AbstractProduct.
- Product.
- Client.



ESTRUCTURA


this.productA = this.factory.createProductA();
this.productB = this.factory.createProductB();
Client
1
1
-factory
1
AbstractFactory
+createProductB()
+createProductA()
ProductA1
-productA
1
AbstractProductA
ProductA2
ConcreteFactory1
+createProductA() : ProductA1
+createProductB() : ProductB1
ProductB1
-productB
1
AbstractProductB
ProductB2
ConcreteFactory2
+createProductA() : ProductA2
+createProductB() : ProductB2

```python
from __future__ import print_function

from abc import ABCMeta, abstractmethod

class Button:
    __metaclass__ = ABCMeta

    @abstractmethod
    def paint(self):
        pass

class LinuxButton(Button):
    def paint(self):
        return "Render a button in a Linux style"

class WindowsButton(Button):
    def paint(self):
        return "Render a button in a Windows style"

class MacOSButton(Button):
    def paint(self):
        return "Render a button in a MacOS style"

class GUIFactory:
    __metaclass__ = ABCMeta

    @abstractmethod
    def create_button(self):
        return Button
```

```python
class LinuxFactory(GUIFactory):
    def create_button(self):
        return LinuxButton()

class WindowsFactory(GUIFactory):
    def create_button(self):
        return WindowsButton()

class MacOSFactory(GUIFactory):
    def create_button(self):
        return MacOSButton()

appearance = "linux"

if appearance == "linux":
    factory = LinuxFactory()
elif appearance == "osx":
    factory = MacOSFactory()
elif appearance == "win":
    factory = WindowsFactory()
else:
    raise NotImplementedError(
        "Not implemented for your platform: {}".format(appearance)
    )

if factory:
    button = factory.create_button()
    result = button.paint()
    print(result)
```

VENTAJAS

- ➔ Aísla las clases concretas.
- ➔ Facilita el intercambio de familias de productos.
- ➔ Promueve la consistencia entre productos: un cliente utiliza sólo una familia de productos a la vez.

DESVENTAJAS

- ★ Es difícil dar cabida a nuevos tipos de productos, ya que se debe modificar la clase abstracta y todas las subclases para soportarlo.

FUENTES DE INFORMACIÓN

- Basham, B., Sierra, K. and Bates, B. (2008). Head first servlets & JSP. Sebastopol (Calif.): O'Reilly, pp.109-168.
- Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (2011). Patrones de diseño. [Madrid]: Pearson Addison Wesley, pp.79-88.