Tecnológico Nacional de México campus Colima

Maestría en Sistemas Computacionales

Patrones de diseño

Ejercicio Práctico: Patrón de Diseño Factory

Docente:

D. en C. Patricia Figueroa Millan

Autor:

Ing. Gilberto Rene Martinez Gutierrez G2146013

Villa de Álvarez, Colima, México

28 de septiembre de 2022

Objetivo

Desarrollar un ejercicio sobre los usos e implementacion del patron de diseño Factory

Metodologia

Para la resolucion del problema se realizo un analisis previo sobre el funcionamiento del patron de diseño factory donde se recurrio al material proporcionado por la profesora, el contenido de este material facilito el entendimiento de lo basico sobre el patron, ademas de permitir analisar unos ejemplos de su funcionamiento

Antecedentes del patron de diseño factory

El patrón de diseño Factory se centra en agregar un nivel extra de abstracción entre la creación del objeto y dónde es éste utilizado, proporciona opciones extras que se pueden extender fácilmente en un futuro. ¿Qué significa esto? que la Fabrica (Factory) se enfoca en la creación de esa abstracción extra entre la creación del objeto y dónde se utiliza. Es uno de los patrones de diseño más sencillos de comprender, permitiendo seleccionar dinámicamente que clases instanciar con base en cierto tipo de lógica. Por ejemplo: Suponga un juego para diseñar casas y la casa tiene un silla que ya está por defecto agregada en el piso de la casa. Al agregar el patron de diseño Factory se le puede proporcionar al usuario la opción de seleccionar dirferentes sillas, y qué tantas sillas en tiempo de ejecución, en lugar de que la silla esté codificada en el proyecto cuando se ejecuta, el usuario tiene ahora la opción de elejir.

Agregar esta abstracción adicional también significa que:

• Las complicaciones de crear instancias de objetos adicionales se puedan ocultar de la clase o del método que lo está utilizando.

• La separación hace que el código sea más fácil de leer y documentar.

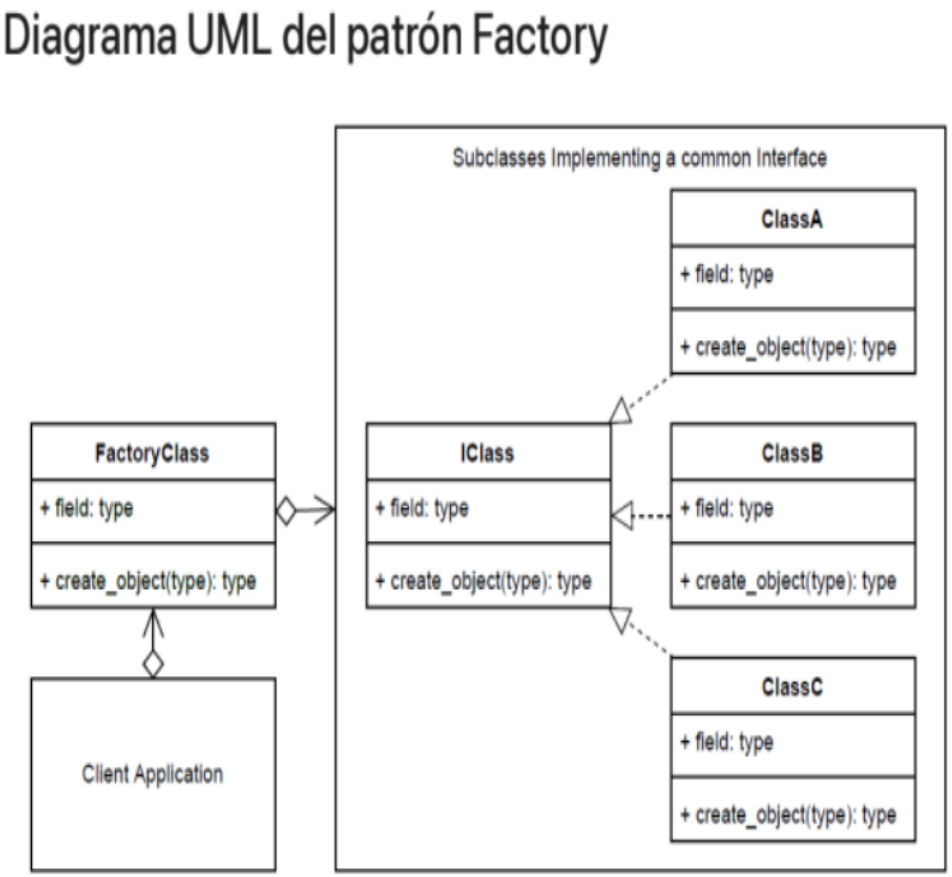
Terminología

•Creador concreto: Cualquier código (aplicación, clase o método que crea objetos), conocido comúnmente como cliente, el cual manda llamar al Creador (Factory)

• Interface del producto: Inteface que describre los atributos y métodos que el Creador requerira con la finalidad de crear el producto u objeto.

• Creador: La clase Factory. Declara el método para fabricar los objetos y la cual regresará el bojeto solicitado. El Creador Concreto puede acceder al Creador.

• El objeto regresado desde la clase Factory.



Instrucciones del ejemplo

• En este ejemplo, el cliente desea crear un objeto llamado b .

• En lugar de crear directamente b en el cliente, este pregunta al creador (fabrica) por el objeto en su luagar.

• El creador o fábrica encuentra la clase relevante usando cierto tiepo de lógica para los atributos de la petición.

• El creador entonces pregunta a la subclase para instanciar el nuevo objeto que regresará como referencia al cliente preguntando por éste.

In [ ]:
```
"Ejemplo Conceptual del Patrón de Diseño Factory"

from abc import ABCMeta, abstractmethod
#Se crea la interfaz producto (IProducto)
class IProducto(metaclass=ABCMeta): "Una clase tipo interfaz hipotética (Producto)"
```

```
@staticmethod
@abstractmethod
def crear_objeto():"Un método abstracto de la interfaz"

#Se crean los "creadores concretos" que llaman al Creador (Fábrica)
class CreadorConcretoA(IProducto):
    "Una clase creador concreto que implementa la interfaz IProducto"

    def __init__(self):
        self.name = "CreadorConcretoA"

    def crear_objeto(self):
        return self

class CreadorConcretoB(IProducto):
    "Una clase creador concreto que implementa la interfaz IProducto"

    def __init__(self):
        self.name = "CreadorConcretoB"

    def crear_objeto(self):
        return self

class CreadorConcretoC(IProducto):
    "Una clase creador concreto que implementa la interfaz IProducto"

    def __init__(self):
        self.name = "CreadorConcretoC"

    def crear_objeto(self):
        return self

# Clase Factory
class Creador:
     "La clase Creador - Factory"
     @staticmethod

     def crear_objeto(propiedad):
            "Método estático para obtener un producto concreto"
            if propiedad == 'a':
                return CreadorConcretoA()
            if propiedad == 'b':
                return CreadorConcretoB()
            if propiedad == 'c':
                return CreadorConcretoC()

            return None
        # Cuando se desea un nivel mayor de abstracción
        # Aplicación cliente
        #PRODUCTO = Creador.crear_objeto('c')
        #print(PRODUCTO.name)
        #Se puede realizar directamente

PRODUCTO = CreadorConcretoB()
print(PRODUCTO.name)
```

```
CreadorConcretoB
```

Problematica del ejemplo

Supongamos que se desea desarrolla un juego de carreras y la tarea es crear tráfico aleatorio en las pistas de carreras, se necesita producir un Carro, una Bicicleta y una Camioneta de manera aleatoria ¿Cómo podemos crear estos objetos?

• 1. Crear un clase para cada tipo de objeto.

• 2. Crear una fabrica de vehiculos.

Pero... ¿Qué pasa si tengo que hacer adecuaciones? ¿cuantas clases y cuantos objetos se verían afectadas?

Solucion

Para crear objetos de las clases mencionadas anteriormente, se empleará el patrón de diseño Factory, el cual a través de un metodo nos devolverá los objetos creados acorde a cada tipo de clase o cualquier clase que se pueda agregar de vehículos.

In [ ]:
```python
from abc import ABCMeta, abstractmethod
class IVehiculo(metaclass=ABCMeta):
    "Una clase tipo interfaz para vehículos (Producto)"
    @staticmethod
    @abstractmethod
    def crear_objeto():
     "Un método abastracto de la interfaz"
```

In [ ]:
```python
class Carro(IVehiculo):
    def __init__(self):
        self.name="Esto es un carro!!"
        self.area = 100

    def crear_objeto(self):
        return self
```

In [ ]:
```python
class Bicicleta(IVehiculo):

#Se agregó un parámetro de ejemplo en el constructor (inicializador)
#Que se pasa a la clase desde el Factory (Creador)
    def __init__(self,tipo):
        self.name="Esta es una bicicleta!!"
        self.area = 50
        self.pedales = True
        self.asiento = True
        self.tipo_bike = tipo
#print(tipo)
    def crear_objeto(self):
        return self
```

In [ ]:
```python
class Camioneta(IVehiculo):
    def __init__(self):
        self.area = 1000
        self.name="Esta es una Camioneta!!"
        self.doble_traccion = False
```

```
    def crear_objeto(self):
        return self
    def velocidad(self):
        return "Arrancando...."
```

In [ ]:
```
class CreadorVehiculos:
    "Factory Class"
    @staticmethod
    def crear_objeto(propiedad):
        "A static method to get a concrate product"
        if propiedad =="carro":
            return Carro()
        elif propiedad == "bicicleta":
            return Bicicleta("montaña")
        elif propiedad == "camioneta":
            return Camioneta()
        else:
            return None
```

In [ ]:
```
# Un objeto se crea normalmente así:

#carro = Car()
#print(carro.name)
#print(carro.area)
# The Client
# Creación de objeto mendiante el factory.
camioneta = CreadorVehiculos().crear_objeto("camioneta")
print(camioneta.name)
print(camioneta.velocidad())
print(camioneta.doble_traccion)
```

```
Esta es una Camioneta!!
Arrancando....
False
```

In [ ]:
```
carro = CreadorVehiculos().crear_objeto("carro")

print(carro.name)
```

```
Esto es un carro!!
```

In [ ]:
```
bike = CreadorVehiculos().crear_objeto("bicicleta")

print(bike.name)
print(bike.pedales)
print(bike.tipo_bike)
```

```
Esta es una bicicleta!!
True
montaña
```

In [ ]:
```
bike2 = CreadorVehiculos().crear_objeto("bicicleta")

print(bike2.name)
print(bike2.pedales)
```

```
Esta es una bicicleta!!
True
```

Ejercicio practico

Suponga un caso de uso de ejemplo, en donde una interfaz de usuario permite al usuario seleccionar elementos de un menú, como sillas.

Al usuario se le ha dado una opción usando algún tipo de interfaz de navegación, y no se sabe qué opción o cuántas seleccionará el usuario hasta que la aplicación se esté ejecutando realmente y el usuario comience a usarla.

Entonces, cuando el usuario seleccione la silla, la fábrica toma alguna propiedad relacionada con esa selección, como un ID, tipo u otro atributo, y luego decide qué subclase relevante instanciar para devolver el objeto apropiado.

In [ ]:
```python
from abc import ABCMeta, abstractmethod
class Interfaz_Sillas(metaclass=ABCMeta):
    "Una clase tipo interfaz para sillas (Producto)"
    @staticmethod
    @abstractmethod
    def crear_objeto():
     "Un método abastracto de la interfaz"
```

In [ ]:
```python
#Subclases de la clase interfaz_sillas
class Silla_Comedor(Interfaz_Sillas):
    def __init__(self):
        self.name="Esto es una silla de comedor!!"
        self.price = 1000
        self.color = "Blanco"
        self.material = "Madera"
        self.patas = 4
        self.tipo = "Comedor"

    def crear_objeto(self):
        return self
```

In [ ]:
```python
class Silla_Exterior(Interfaz_Sillas):
    def __init__(self):
        self.name="Esto es una silla de exterior!"
        self.price = 1500
        self.color = "Blanco"
        self.material = "Metal"
        self.patas = 4
        self.tipo = "Exterior"

    def crear_objeto(self):
        return self
```

In [ ]:
```python
class Silla_Periquera(Interfaz_Sillas):
    def __init__(self):
        self.name="Esto es una periquera !!"
        self.price = 800
        self.color = "Blanco"
        self.material = "Madera"
        self.patas = 3
        self.tipo = "Periquera"
```

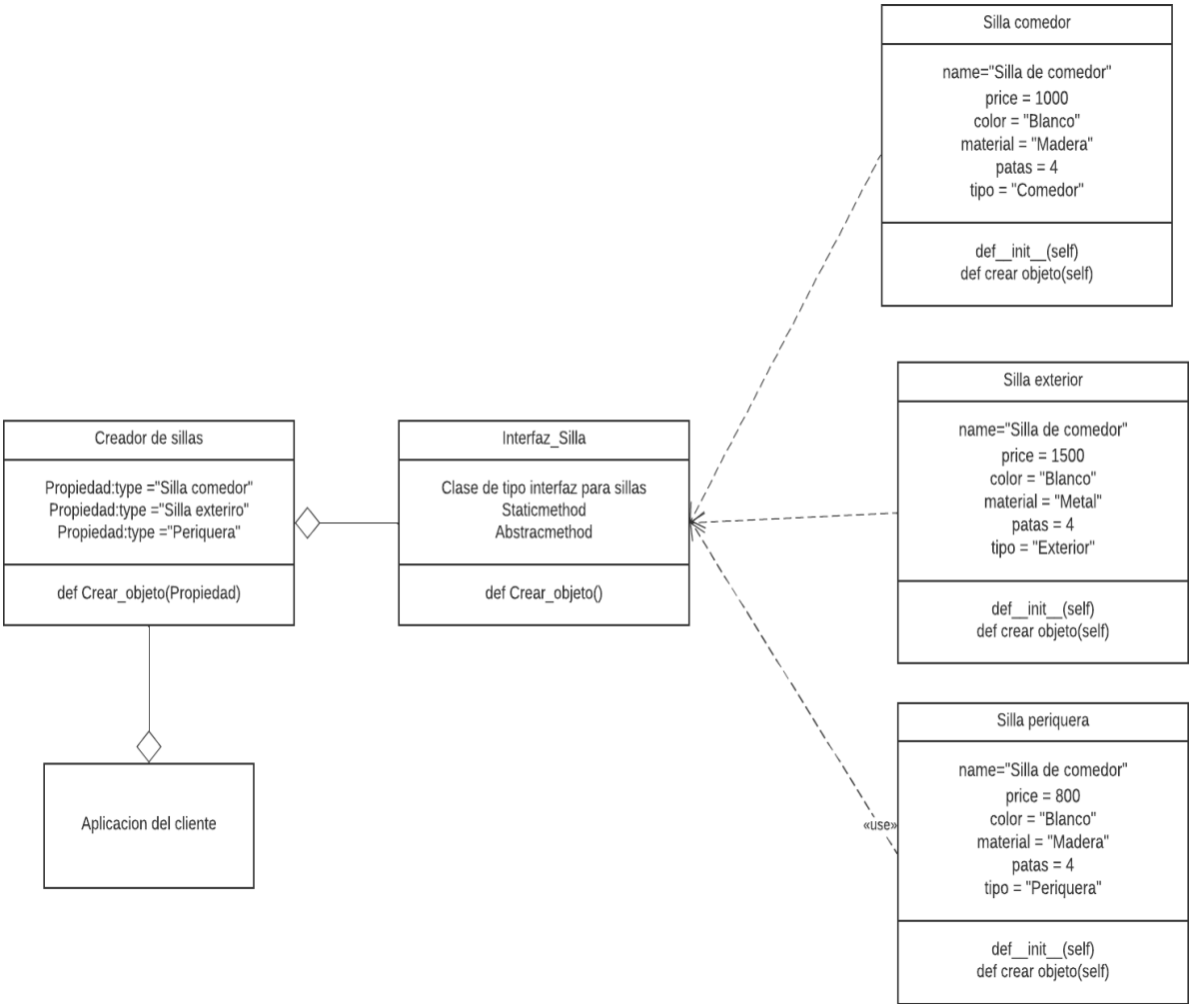
```
    def crear_objeto(self):
        return self
```

In [ ]:
```
# Clase constructora
class CreadorSillas:
    "Factory Class"
    @staticmethod
    def crear_objeto(propiedad):
        "Metodo estatico para la contruccion de objetos"

        if propiedad =="Silla comedor":
            return Silla_Comedor()

        elif propiedad == "Silla exterior":
            return Silla_Exterior()

        elif propiedad == "Periquera":
            return Silla_Periquera()

        else:
            return None
```

In [ ]:
```
# Creacion de objetos
Id_1506_Silla = CreadorSillas().crear_objeto("Silla comedor")
print(Id_1506_Silla.name)
print(Id_1506_Silla.price)
print(Id_1506_Silla.color)
print(Id_1506_Silla.material)
print(Id_1506_Silla.patas)
print(Id_1506_Silla.tipo)
```

```
Esto es una silla de comedor!!
1000
Blanco
Madera
4
Comedor
```

Diagrama UML del programa sillas



Conclusiones

Gracias al desarrolo de esta actividad se logro comprender el funcionamiento del patron de diseño "Factory", a traves de este patron se comprendio la estructura con la cual se deben diseñar sistemas expansibles dentro de la creacion de objetos