

Diseño y Arquitectura De Software

Patrón de Diseño “FACTORY”

Lic. Ángel Santiago Jaime Zavala

Alumna: Claudia Abelina Vázquez
Seca



La ingeniería del software propone metodologías a aplicar en todo el proceso de creación del software que no es solo programación, pero esa es la etapa donde los patrones de diseños actúan.

Los Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características.

- Una de ellas es que se debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores.
- Otra es que debe ser reutilizable, lo que significa es que es aplicable a diferentes problemas de diseño en distintas circunstancias.

“Los patrones de diseño son recetas”

Probadas por muchos desarrolladores y que han dado fe de que funcionan para determinados problemas, solo debemos aprender a usarlos en los contextos adecuados, ya que dependiendo del problema elegiremos uno u otro.

Factory Pattern (patrón de fabrica)

Factory Method (patrón de fabrica)

El patrón Factory hace parte de los patrones de creación.

Dentro de los patrones de diseño existen tres categorías:

Patrones de creación.

Patrones de comportamiento.

Patrones estructurales.

El patrón de fabrica, centraliza en una clase la creación de objetos de un subtipo común **determinado**, ocultando al usuario la instancia del objeto solicitado.







Como su nombre lo indica, la idea de este patrón es tener una fábrica que cree objetos de distinto tipo, esto es supremamente útil cuando no se sabe con antelación que objeto crear, por lo tanto se crearán en tiempo de ejecución. La factoría hace uso de parámetros para determinar qué objeto debe crear, donde nosotros debemos de proporcionarle tales parámetros.

Ingredientes:

- 1.Una super-clase llamada Persona.
- 2.Dos clases que llevarán por nombre Masculino y Femenino, estas heredan de Persona.
- 3.Una clase Factoría, que nos representará la factoría como tal.
- 4.Un pequeño fichero, al que llamaremos main desde donde se iniciará la aplicación.
- 5.Sal y pimienta al gusto :)

Preparación:

En la siguiente imagen podrás ver los ficheros que he creado, todos en el mismo nivel.

Name	Date modified	Type	Size
 persona	27/12/2015 06:54 ...	Python File	1 KB
 factoria	27/12/2015 06:53 ...	Python File	1 KB
 femenino	27/12/2015 06:52 ...	Python File	1 KB
 masculino	27/12/2015 06:52 ...	Python File	1 KB
 main	27/12/2015 06:51 ...	Python File	1 KB
 __init__	27/12/2015 05:45 ...	Python File	0 KB

Propósito:

Definir una interfaz para la creación de un objeto, pero permitiendo a las subclases decidir de que clase instanciarlo. Permite, por tanto, que una clase difiera la instanciación en favor de sus subclases.

Porque utilizar el Patrón de Diseño Factory:

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.

Cuando utilizar el Patrón de Diseño Factory:

- Cuando una clase no puede adelantar las clases de objetos que debe crear.
- Cuando una clase pretende que sus subclases especifiquen los objetos que ella crea.
- Cuando una clase delega su responsabilidad hacia una de entre varias subclases auxiliares y queremos tener - localizada a la subclase delegada.

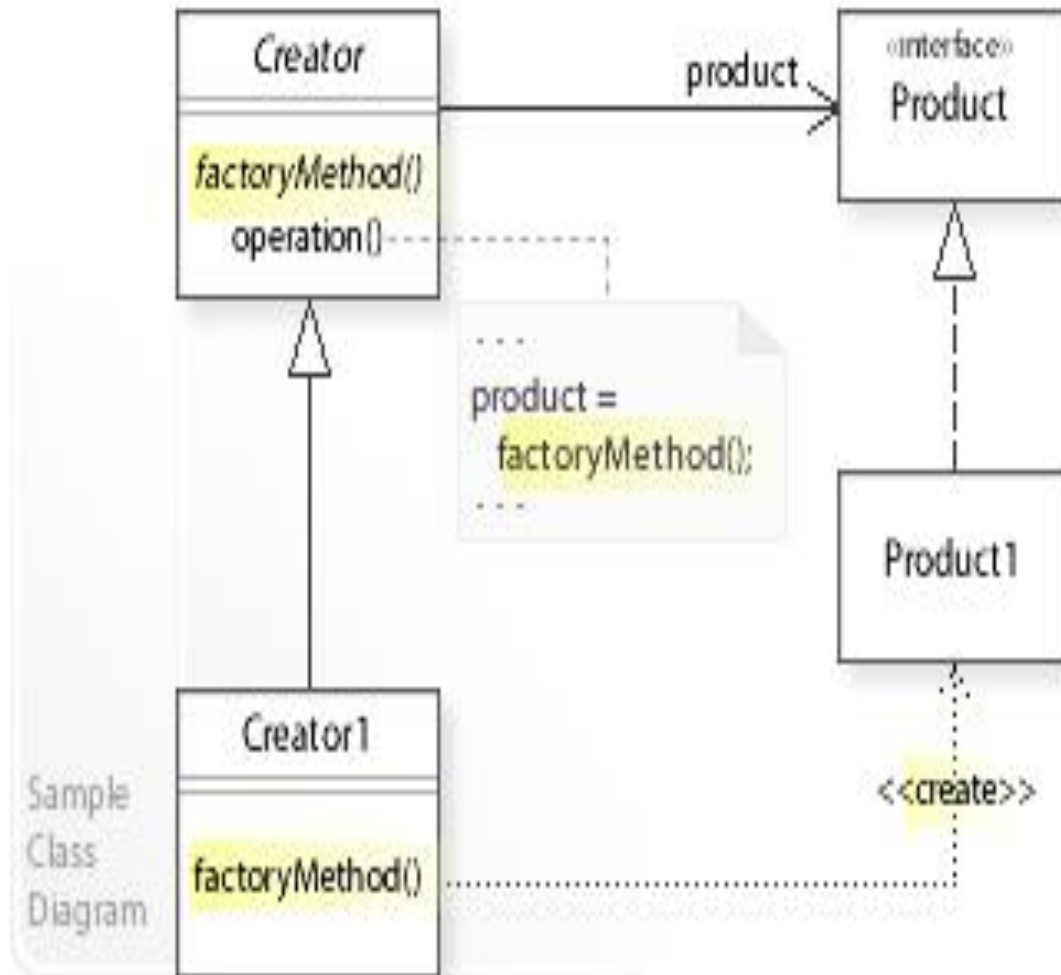
Ventajas:

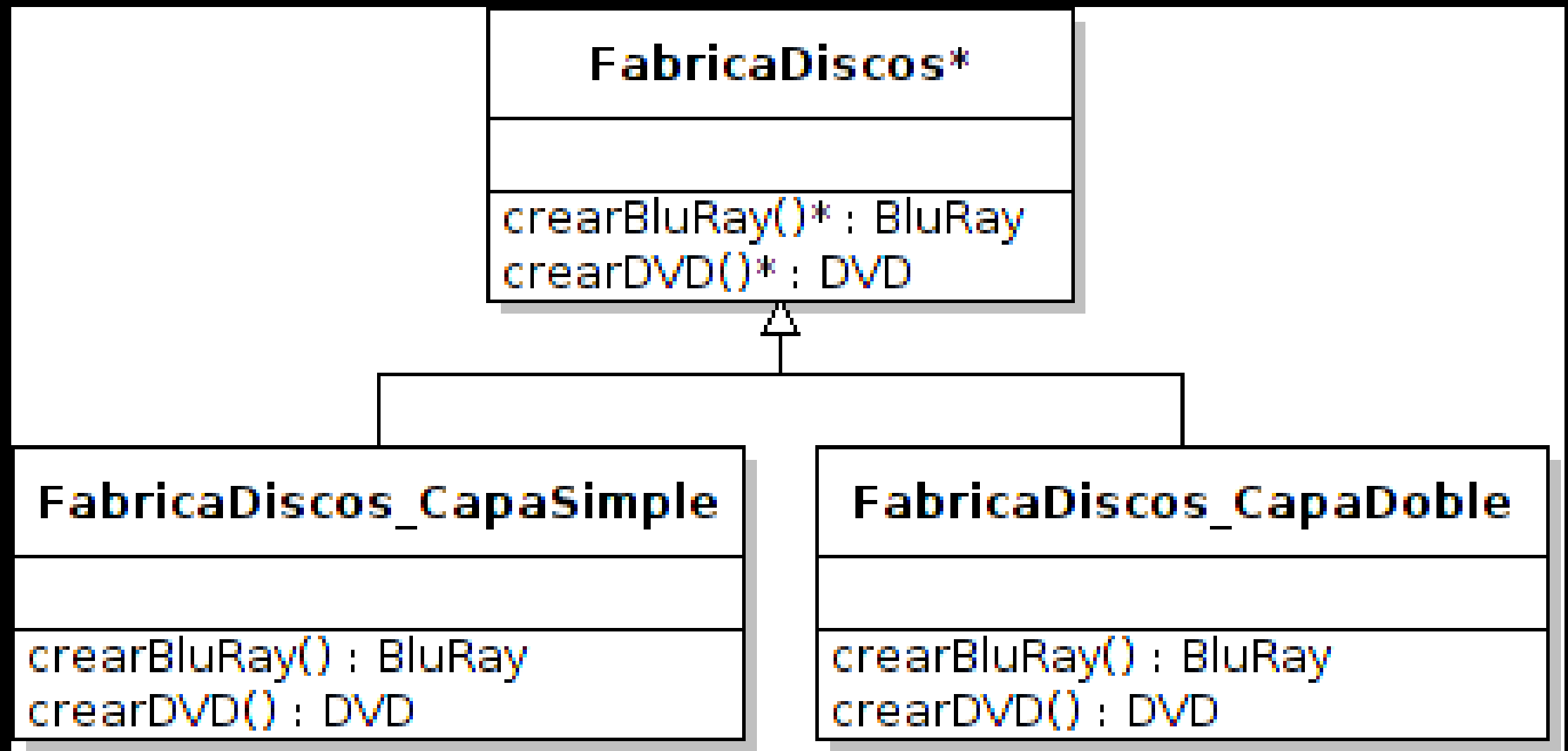
- Se gana en flexibilidad, pues crear los objetos dentro de una clase con un método es siempre más flexible que hacerlo directamente, debido a que se elimina la necesidad de atar nuestra aplicación unas clases de productos concretos.
- Se facilitan futuras ampliaciones, puesto que se ofrece las subclases la posibilidad de proporcionar una versión extendida de un objeto, con sólo aplicar en los Productos la misma idea del método.
- Se facilita, en cuanto a que se hace natural, la conexión entre jerarquías de clases paralelas, que son aquellas que se generan cuando una clase delega algunas de sus responsabilidades en una clase aparte. Ambas jerarquías de clases paralelas son creadas por un mismo cliente y el patrón establece la relación entre parejas de subclases concretas en las mismas.

Participantes del Patrón de Diseño Factory:

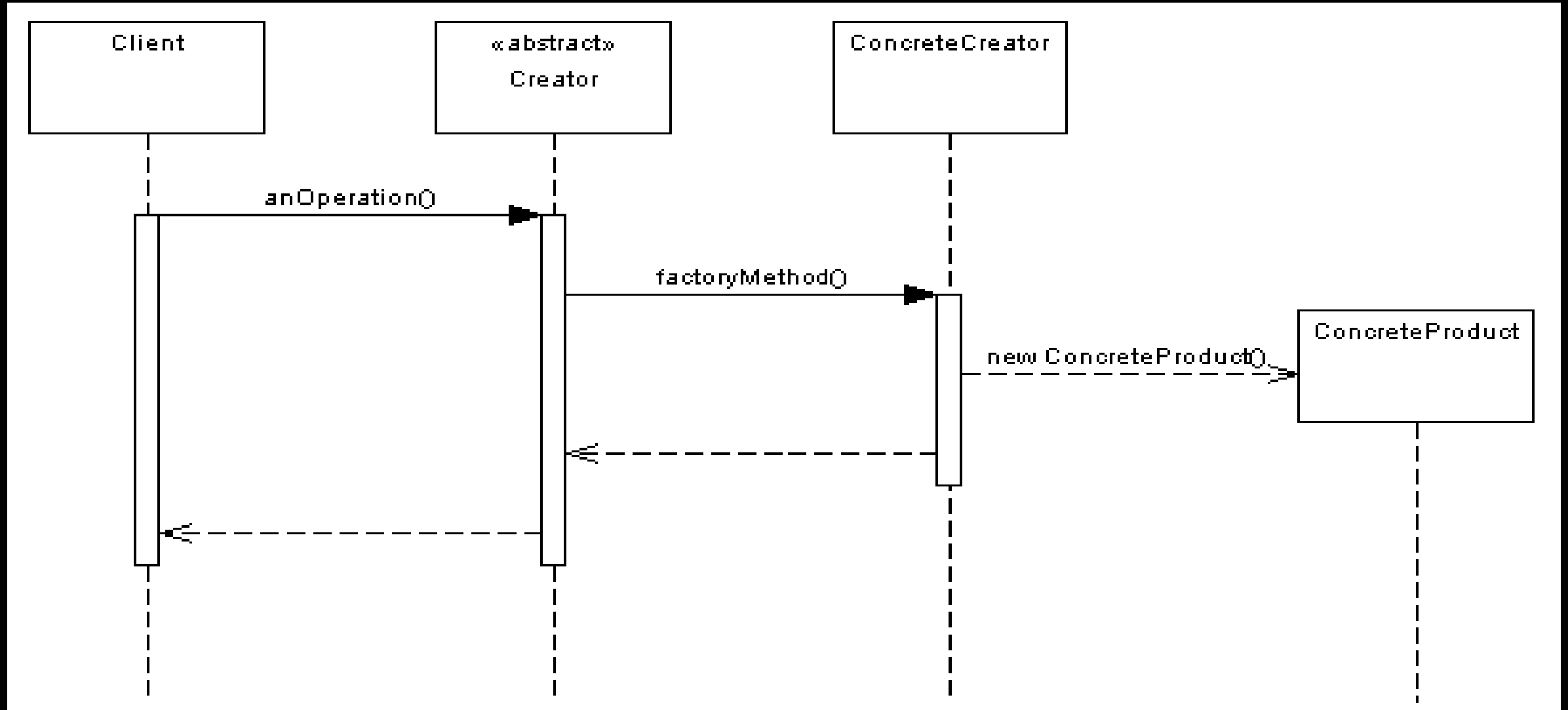
- Producto:** Define la interfaz de los objetos que crea el método de fabricación.
- Producto Concreto:** Implementa la interfaz Producto.
- Creador:** Declara el método de fabricación, el cual devuelve un objeto del tipo Producto. También puede definir una implementación predeterminada del método de fabricación que devuelve un objeto Producto Concreto. Puede llamar al método de fabricación para crear un objeto Producto.
- Creador Concreto:** Redefine el método de fabricación para devolver una instancia de Producto Concreto.

UML – Factory Method





El patrón de fabrica se puede llegar a usar bastante si se usa la metodología de programación orientada a objetos.



• Super Clase Persona:

""" Clase que define a una persona """

```
class Persona(object): """Para nuestro caso, una persona tendrá un nombre,  
una edad y un genero, por lo general en Java esta clase sería una 'interfaz' """  
    def __init__(self):  
        self.nombre = None  
        self.edad = None  
        self.genero = None  
        # Algunos getters ...  
    def get_nombre(self):  
        return self.nombre  
    def get_edad(self):  
        return self.edad  
    def get_genero(self):  
        return self.genero  
    def __str__(self):  
        return "Informacion de una persona:\n1. Nombre:  
{n}\n2. Edad: {e}\n3. Genero: {g}".format(n=self.get_nombre(),  
e=self.get_edad(), g=self.get_genero())
```

- Clase Femenino:

```
from persona import Persona
class Femenino(Persona):
    """Esta clase hereda de la super clase Persona, solo
    definiremos su constructor"""

    def __init__(self, nombre, edad, genero):
        self.nombre = nombre
        self.edad = edad
        self.genero = genero
        print "Hola Miss "+nombre+" su edad es
"+str(edad)
```


- Clase Masculino:

```
from persona import Persona
```

```
class Masculino(Persona): # Heredamos de Persona
```

```
"""Esta clase hereda de la super clase Persona, solo  
definiremos su constructor"""
```

```
def __init__(self, nombre, edad, genero):  
    self.nombre = nombre  
    self.edad = edad  
    self.genero = genero  
    print "Hola mister "+nombre+" su edad es "  
        +str(edad)
```

- Clase Factoria:

```
from femenino import Femenino
from masculino import Masculino
```

```
class Factoria(object):
```

```
    """Esta clase es nuestra factoria, como ya sabes Python
define un constructor sin argumentos por default para cada
clase, por eso no hace falta escribir uno"""
```

```
    def get_persona(self, nombre, genero, edad):
```

```
    """Metodo que retorna objetos persona según el genero"""
```

```
        #genero es el parametro usado por la factoria #para
elegir    el obj a crear
```

```
        if (genero is 'F'):
```

```
            return Femenino(nombre, edad, genero)
```

```
        elif (genero is 'M'):
```

```
            return Masculino(nombre, edad, genero)
```

- Fichero main:

```
import factoria

if __name__ == '__main__':
    mi_factoria = factoria.Factoria()

    #Factoria, crea a una persona! persona =
    mi_factoria.get_persona('Guido Vann Rosum',
        'M', 30)
    #se ha creado una persona masculina
    print persona
    # print persona.get_nombre()
    # print persona.get_genero()
```

Bibliografía

<https://github.com/tectijuana/pypdd6a-Archived/wiki/Patr%C3%B3n-Factory-Python>

<http://carmoreno.github.io/python/2016/01/06/Patron-Factory/>

<http://fedora.soup.io/post/305304997/Patrones-de-Dise-o-en-Python-Factory>



GRACIAS!!!