
| | | |
|---|---|---|
| **Facultad** | **:** | Tecnología |
| **Curso** | **:** | Desarrollo de Aplicaciones Móviles |
| **NRC** | **:** | 1725 |
| **Tema** | **:** | Patrones Factory |
| **Profesor** | **:** | Gustavo Coronel Castillo |
| **Periodo** | **:** | 2017 - 1 |
| **Alumno** | **:** | Jeremy Tornero Landeo |

**Miraflores, 27 de junio del 2017**

# Introducción

Todos aquí programamos.

Bueno, hacemos el intento.

Cada uno de nosotros utiliza utiliza su propia lógica, conocimientos y experiencia para crear código. Pero...

¿Qué pasaría si estamos trabajando y tenemos que compartir nuestro código con un compañero?

¿Es posible desarrollar un módulo que otro programador pueda **aprovechar**, **entender** y **mejorar**?

Sí

## Patrones de diseño

¿Patrones?

"Cada patrón describe un problema que ocurre una y otra vez en nuestro medio ambiente y, a continuación describe el núcleo de la solución a ese problema, de tal manera que se puede utilizar esta solución un millón de veces, sin tener que hacerlo de la misma manera dos veces".

*Christopher Alexander y Murray Silverstein*
*(A Pattern Language, 1977)*

Reglas

Principios

**Estudio**

## Entonces...

- Es una forma reutilizable de resolver un problema común.
- Se enfoca en la solución, no en el problema.
- No es un paradigma de programación.
- No depende de un lenguaje en específico.

## ¿Y los patrones de diseño?

Refiere al diseño de interacción o interfaces.


Nota: También existen otras categorías de patrones.

## Ventajas

- Estandarizar el lenguaje entre programadores.
- Evitar perder tiempo en soluciones a problemas ya resueltos o conocidos.
- Crear código reusable.

## Tipos

- Creacionales. **Fabricación** →
- Estructurales.
- Comportamiento.

**Simple**

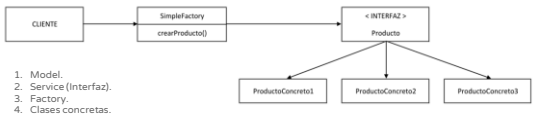
**Method**

**Abstract**

# Desarrollo

## Simple Factory

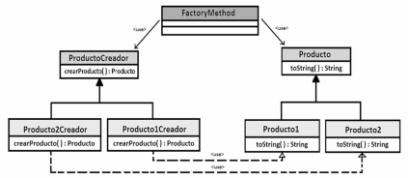
Tenemos una clase de fábrica que tiene un método que devuelve un tipo de objeto según lo que decida el cliente. Puede evolucionar a Method o Abstract.




1. Model.
2. Service (Interfaz).
3. Factory.
4. Clases concretas.
5. View.
6. Prueba.

## Factory Method

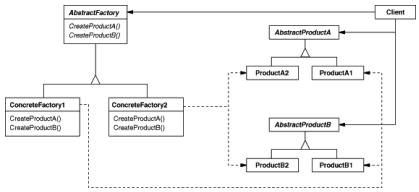
Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue sus subclases la creación de objetos.




1. Model.
2. Factory Method..
3. Factory.
4. Productos.
5. View.
6. Prueba.

## Abstract Factory

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




1. Model.
2. Factory.
3. Factory 2.
4. Producto.
5. View.
6. Prueba.

# Conclusiones

- La idea principal es optimizar nuestro código, hacerlo más limpio y escalable.
- El reto es aprender a construir código eficiente, bajo buenas prácticas.
- El trabajo colaborativo debe ser más fácil.
- Aplicar estas técnicas facilita la estructura, construcción y mantenimiento del sistema.

# Bibliografía

Italo Moráles. (2015). ¿Qué son y para qué sirven los patrones de diseño?. 26/06/2017, de Platzi Sitio web: https://platzi.com/blog/patrones-de-diseno/

León Welicki. (2014). Patrones de Fabricación: Fábricas de Objetos. 27/06/2017, de Microsoft Sitio web: https://msdn.microsoft.com/es-es/library/bb972258.aspx