Universidad Autónoma de Entre Ríos

FCyT FACULTAD DE CIENCIA Y TECNOLOGÍA UADER
SEDE CONCEPCIÓN DEL URUGUAY

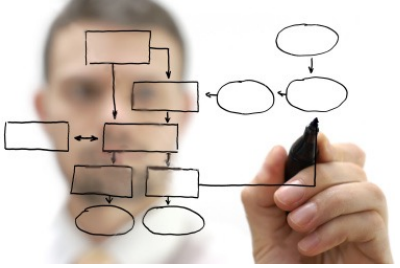
Ingeniería de Software II

Patrones de Diseño

Patrones de Creación

Docente: Dr. Pedro E. Colla

Patrones de diseño



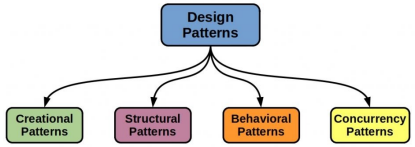
- Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces).

Patrones

Un patrón de diseño es una solución general reusable que puede ser aplicada a problemas que ocurren comúnmente en el desarrollo de software.

- El concepto de patrones de diseño fue el resultado de un trabajo realizado por un grupo de 4 personas Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, (*Gang of four*) que se publicó en los 90s en un libro titulado "*Design Patterns. Elements of Reusable Object-Oriented Software*" en el que se esbozaba (originalmente) 23 patrones de diseño.
- Define una estructura de clases que soluciona un problema particular.
- Debe haber comprobado su efectividad para resolver problemas.
- Debe ser reusable y aplicable en diferentes dominios.

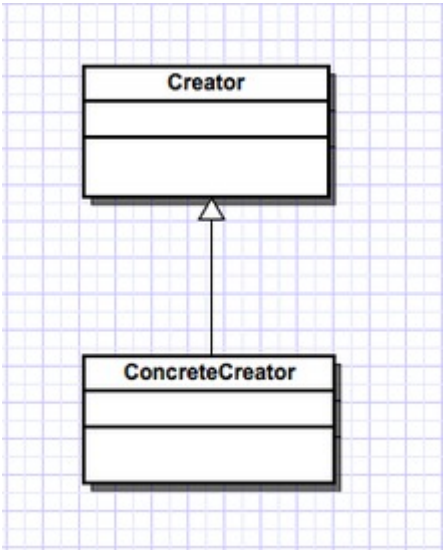
Tipos de patrones



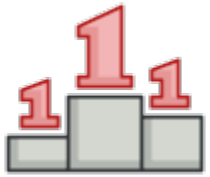
- **Patrones de creación**
  Los patrones de creación abstraen la forma en la que se crean los objetos, permitiendo tratar las clases a crear de forma genérica dejando para más tarde la decisión de qué clases crear o cómo crearlas.
- **Patrones estructurales**
  Tratan de conseguir que cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Lo fundamental son las relaciones de uso entre los objetos, y, éstas están determinadas por las interfaces que soportan los objetos.
- **Patrones de comportamiento**
  Los patrones de comportamiento estudian las relaciones entre llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal.

Creación	Estructurales	Comportamiento
Singleton Factory Builder Prototype	Adapter Bridge Composite Decorator Fachade Flyweight Proxy	Chain of responsibility Command Interpreter Iterator Mediator Memento Observer … et al.

Patrones de creación



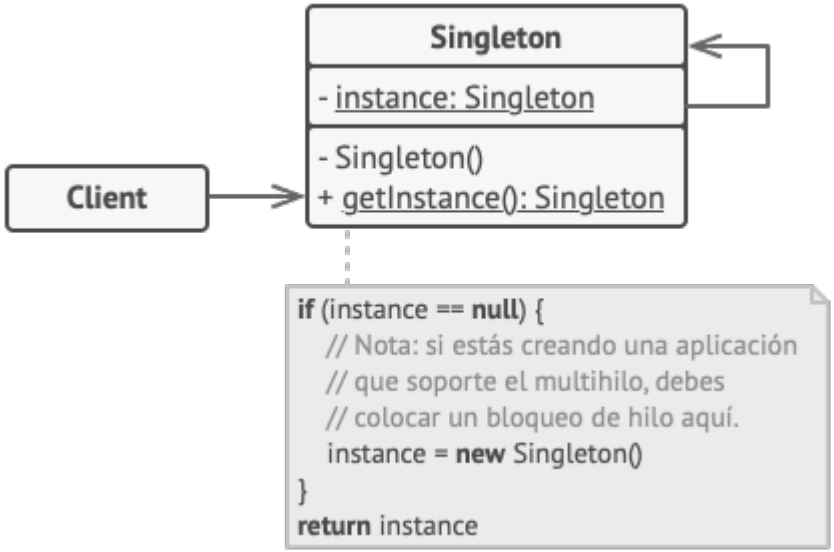
Singleton



Singleton

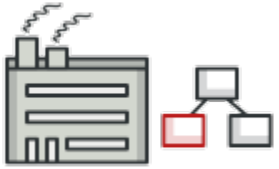
Es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

Propósito	Problema
• Garantizar que una clase tenga una única instancia. • Proporcionar un punto de acceso global a dicha instancia.	Dos funciones en una misma clase (reduce cohesión).



Python3.7 singleton.py

Factory

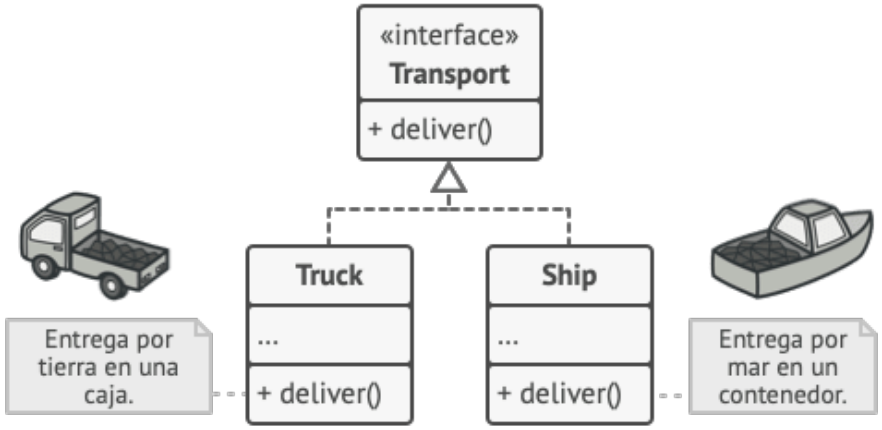


Factory

Es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

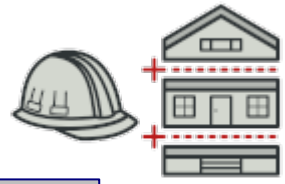
Propósito

- *Factory* es un patrón de diseño creacional que resuelve el problema de crear objetos sin especificar sus clases concretas
- Evita acoplamiento.
- Principio de responsabilidad única. Se puede mover el código de creación de producto a un lugar del programa, haciendo que el código sea más fácil de mantener.
- Principio de abierto/cerrado. Puede incorporar nuevos tipos de productos en el programa sin descomponer el código existente.



python3.7 factory.py

Builder

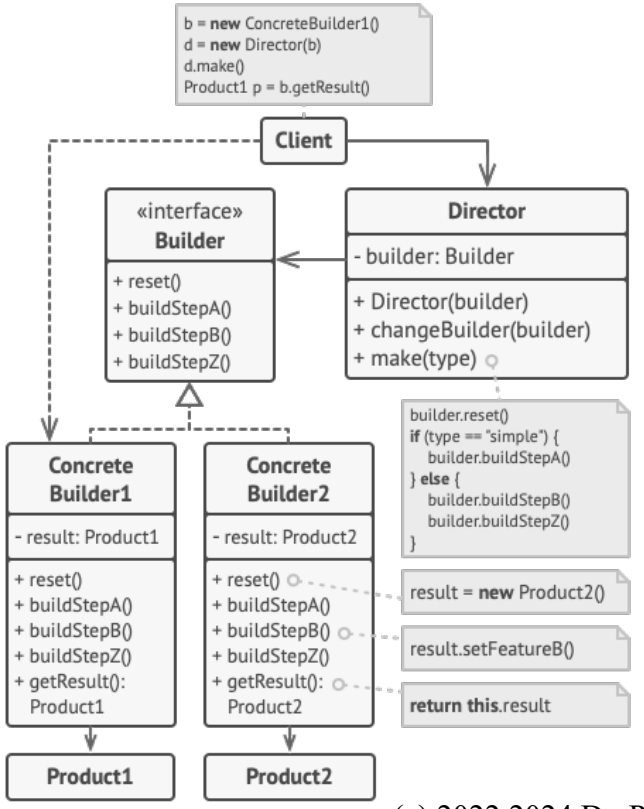


Builder

Es un patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

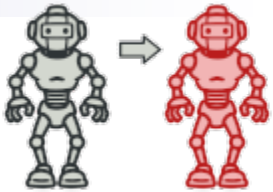
Propósito

- Puede construir objetos paso a paso, aplazar pasos de la construcción o ejecutar pasos de forma recursiva.
- Define el método de construcción, no el comportamiento ulterior de la clase.
- Puede reutilizar el mismo código de construcción al construir varias representaciones de productos.
- Principio de responsabilidad única. Puede aislar un código de construcción complejo de la lógica de negocio del producto.



python3.7 builder.py

Prototype

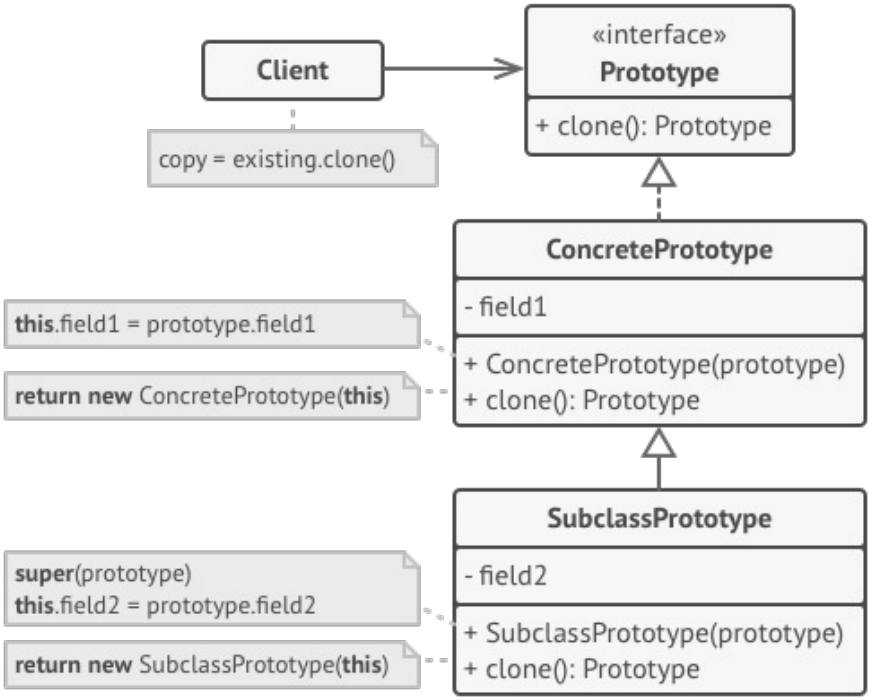


Prototype

Es un patrón de diseños que nos permite copiar objetos existentes sin que el código dependa de sus clases..

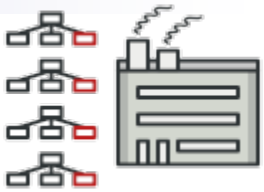
Propósito

- Puede clonar objetos sin acoplarlos a sus clases concretas.
- Puede evitar un código de inicialización repetido clonando prototipos prefabricados.
- Puede crear objetos complejos con más facilidad.
- Obtiene una alternativa a la herencia al tratar con pre-ajustes de configuración para objetos complejos.
- **¡Cuidado con referencias circulares!**



python3.7 prototype.py

Abstract Factory



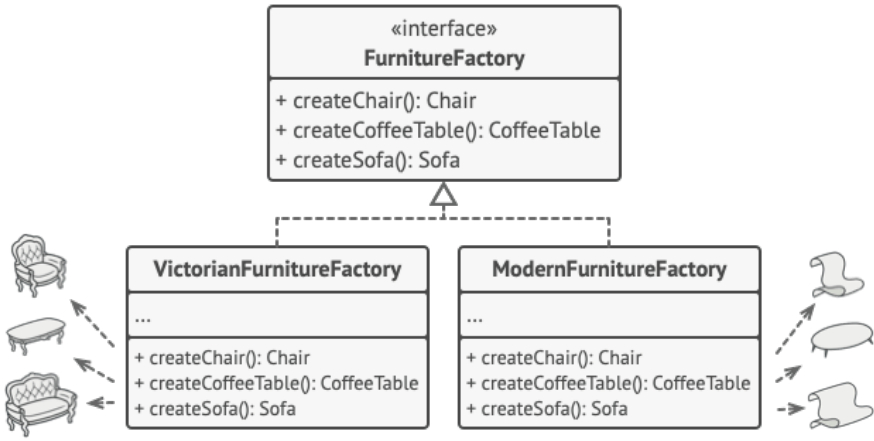
Abstract Factory

Es un patrón de diseño creacional que nos permite producir familias de objetos relacionados sin especificar sus clases concretas.

Propósito

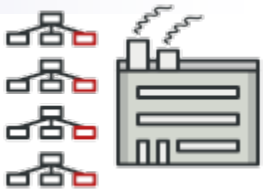
- Tener la certeza de que los productos que obtienes de una fábrica son compatibles entre sí.
- Evitar un acoplamiento fuerte entre productos concretos y el código cliente.
- Principio de responsabilidad única. Puede mover el código de creación de productos a un solo lugar, haciendo que el código sea más fácil de mantener.
- Principio de abierto/cerrado. Puede introducir nuevas variantes de productos sin descomponer el código cliente existente.

- **Evolución de Factory--**AbstractFactory utiliza la composición para delegar la responsabilidad de la creación de un objeto a otra clase mientras Factory utiliza la herencia y se apoya en la clase o subclase derivada para crear el objeto.



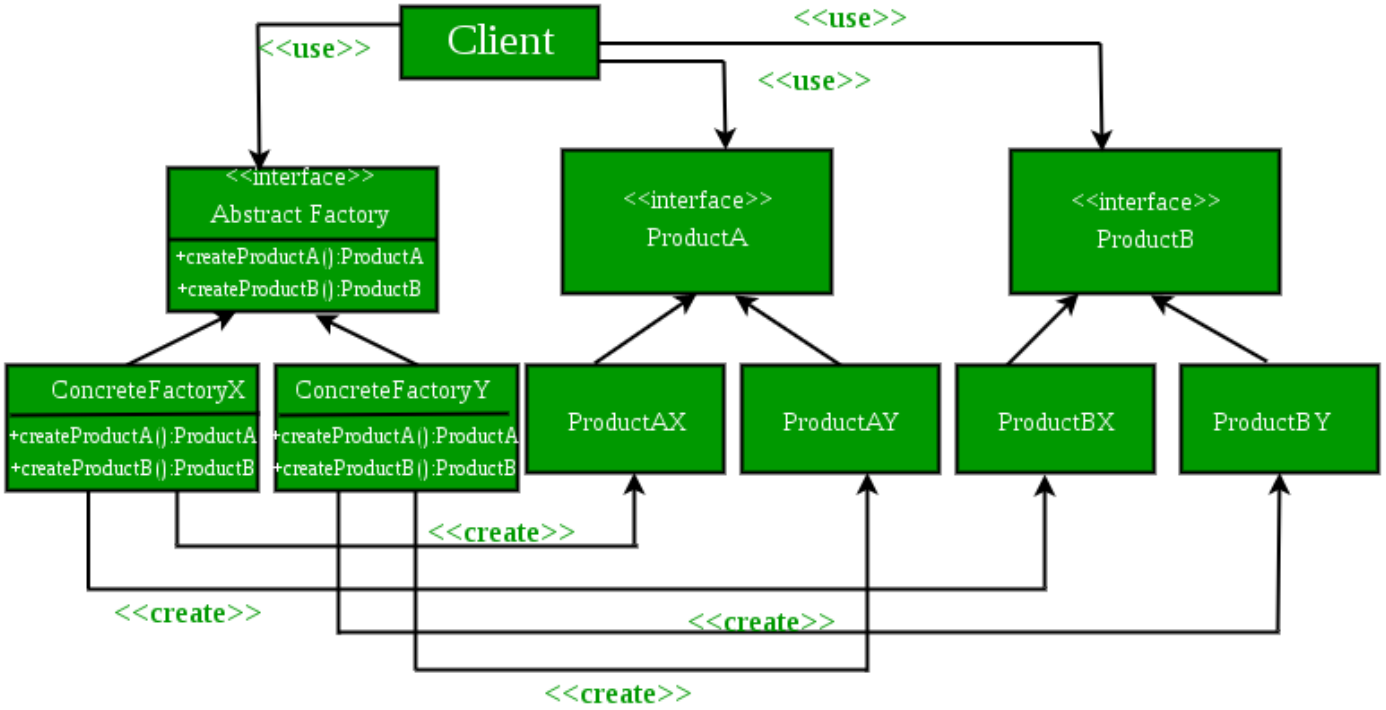
python3.7 abstractfactory.py

Abstract Factory vs. Factory



Abstract Factory vs. Factory

El FACTORY crea objetos concretos mientras que el segundo selecciona FACTORY en "*run-time*". Se utilizan en arquitecturas donde de acuerdo a detalles de contexto las clases FACTORY pueden cambiar (ej. Diferentes geografías requieren crear objetos diferentes).



.py

Taller

- Desarrolle una clase que para una empresa dada devuelva su CUIT (asumido como único). Use como plantilla el patrón llamado "*singleton.py*". (*¿cómo diferenciar referencia de valor?*)

- Desarrolle una clase para implementar un remito que tenga como posibles mecanismos de distribución el correo, mensajería o portal de ventas. Use como plantilla el patrón llamado "*factory.py*". (*¿cómo agregaría "retira por recepción"?*)

- Generar una clase genérica para montar vehículos de modelos distintos a medida que se vayan necesitando. Use como plantilla el patrón llamado "*builder.py*". (*¿Se anima a agregar un nuevo vehículo?*)

- Genere un ejemplo para la creación repetitiva de objetos utilizando un patrón prototipo. (*¿por qué demora 6 segundos en vez de 3 segundos?*)

Referencias en https://refactoring.guru/es/design-patterns/creational-patterns

¿Preguntas?

Bibliografía

- Architecture Patterns with Python – Percival & Gregory – O'Reilly
- Design Patterns – Lasater
- https://refactoring.guru/design-patterns/python