Diseño y Arquitectura de Software.

PATRONES DE DISEÑO

Ana Karen Gpe. Cazares de Leon.

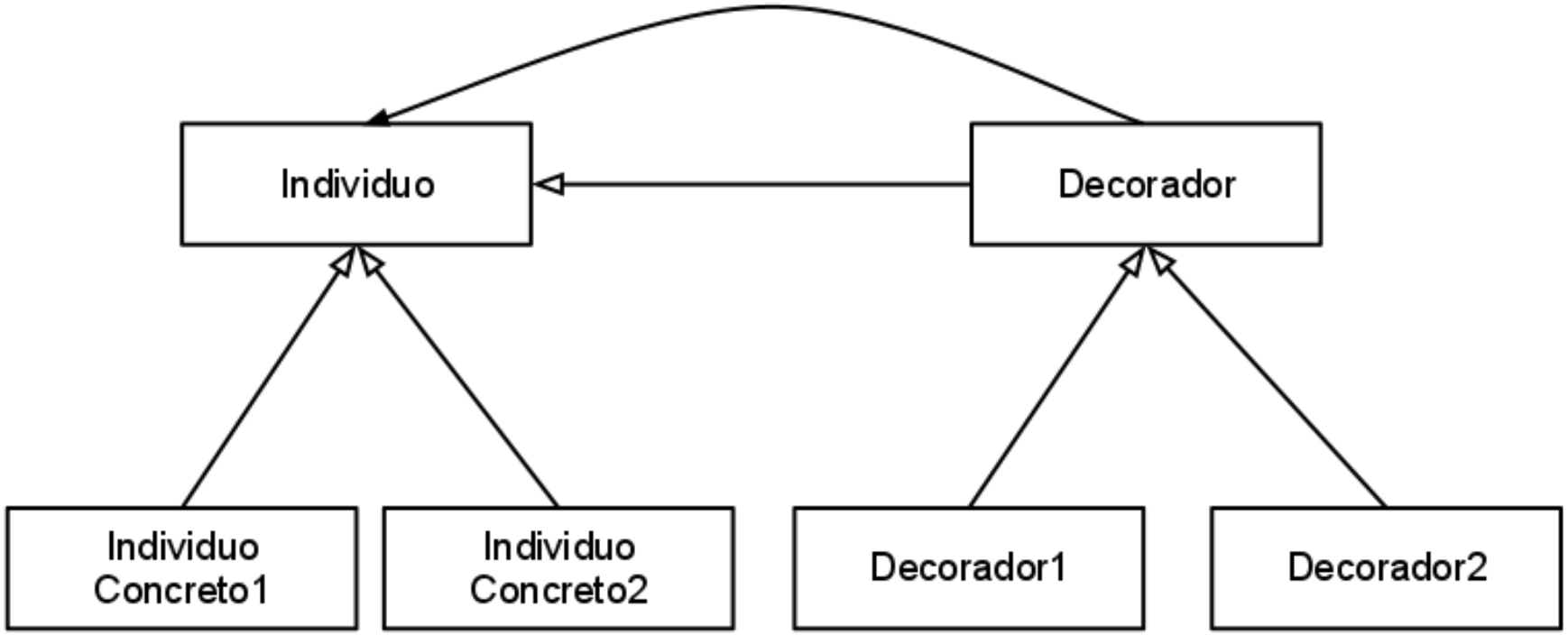
Introducción:

Ana Karen Gpe. Cazares de Leon.

- La herencia extiende el comportamiento de una nueva clase hija ligándola fuertemente con su clase padre.
- A veces, necesitamos un mecanismo más flexible que la herencia para ampliar el comportamiento de una clase. De hecho, se recomienda utilizar la composición frente a la herencia.
- El patrón de diseño **Decorador** nos ofrece una solución para añadir nueva funcionalidad a una clase, más flexible que la herencia, y de manera dinámica.

Decorator:

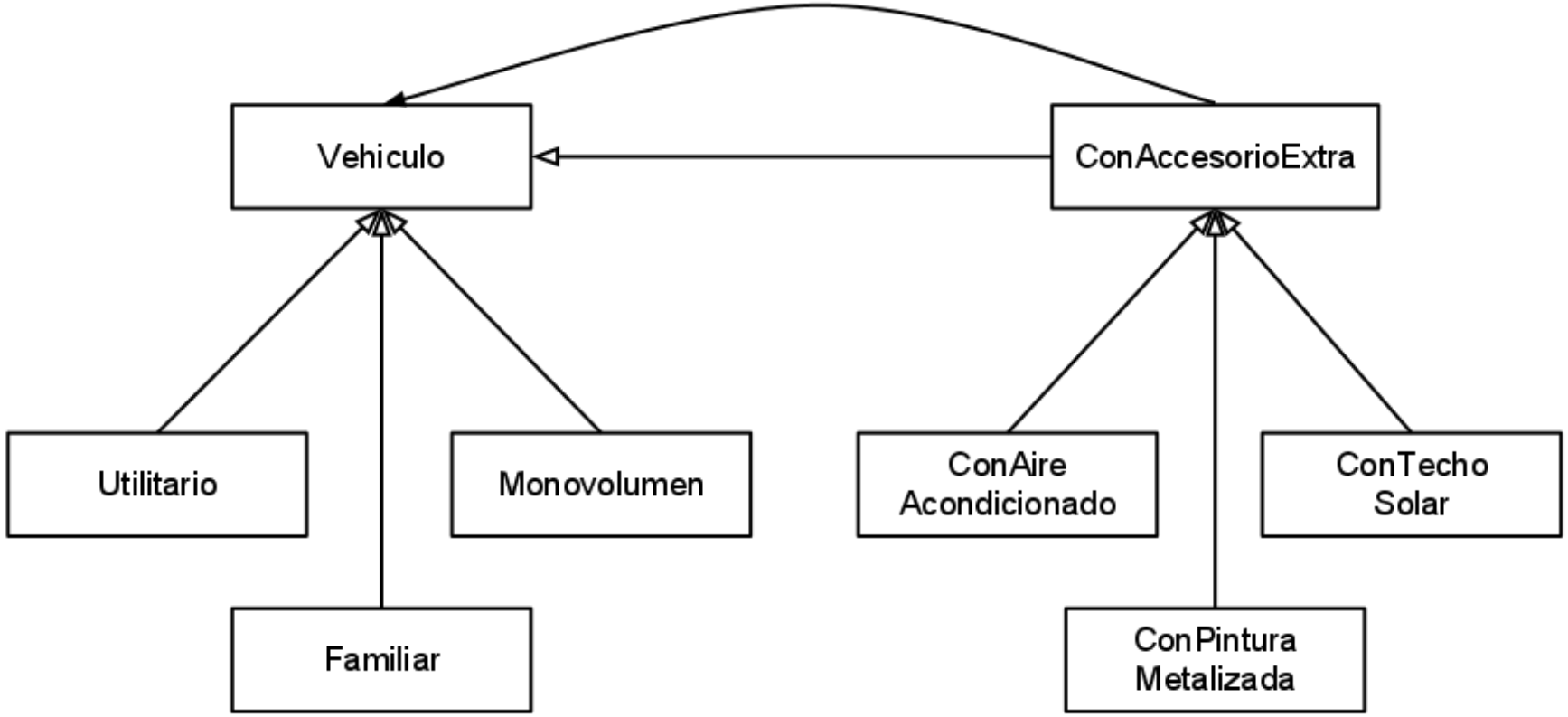
- La utilidad principal del patrón Decorator, es la de **dotar de funcionalidades dinámicamente a objetos mediante composición**. Es decir, vamos a *decorar* los objetos para darles más funcionalidad de la que tienen en un principio.
- Esto es algo verdaderamente útil cuándo queremos **evitar jerarquías de clases complejas**. La herencia es una herramienta poderosa, pero puede hacer que nuestro diseño sea mucho menos extensible.



Este sería el modelo UML general del Patrón Decorador:

Ana Karen Gpe. Cazares de Leon.
2/21/2018

Ana Karen Gpe. Cazares de Leon.
2/21/2018



En un caso concreto:

Ventajas y Desventajas

- **Ventajas:**
- Diseño más flexible que utilizando herencia.
- La funcionalidad se va añadiendo según la vamos necesitando.

- **Desventajas:**
- Desde el punto de vista de la igualdad de objetos, un objeto recubierto y el objeto original no se pueden comparar.
- Si creamos muchos recubridores, puede dar lugar a una explosión de clases.

Decoradores en python.

- Cuando hablamos de los decoradores en python, no se implementa exactamente como lo describe inicialmente el Patrón Decorator, aunque su implementación si es muy similar a la implementación de otros lenguajes tales como c# o java. Un decorador en python permite extraer lógica común e implementarla en una variación de la sintaxis del lenguaje que permite una fácil adición y al mismo tiempo una mejor lectura.
- Es importante entender que la meta no es sobre escribir o modificar la lógica que la clase que se decora, la meta se puede decir es agregar un paso extra a la ejecución de una función o la declaración de un objeto. Permitiendo ejecutar tareas previas o posteriores a la ejecución e inclusive manipular parámetros y el resultado. Sin embargo, la forma de que el decorador se implementa, transfiere la responsabilidad de la ejecución de la función decorada, al decorador mismo. Por lo que no se debe olvidar siempre llamar la función recibida por parámetro que se quiere decorar.
- Los siguientes ejemplos son decoraciones de diferentes tipos, una compilación de varios ejemplos que he encontrado y que me parecen apropiados.

Ejemplo Python:

Interceptando_el_retorno_de_la_funcion.py

Decorador_en_forma_de_funcion.py

Decorador_en _forma_de_clase.py

EncadenandoDecoradores.py

Decoradores_con_argumento.py

Ana Karen Gpe. Cazares de Leon.
2/21/2018

Biografía:

- https://www.genbetadev.com/metodologias-de-programacion/patrones-de-diseno-decorator
- http://www3.uji.es/~belfern/Docencia/Presentaciones/ProgramacionAvanzada/Tema2/decorador.html#2
- https://es.wikipedia.org/wiki/Decorator_(patr%C3%B3n_de_dise%C3%B1o)
- http://www.slothslab.com/python/design%20paterns/2015/12/24/decoradores-python.html