AYUDANTÍA

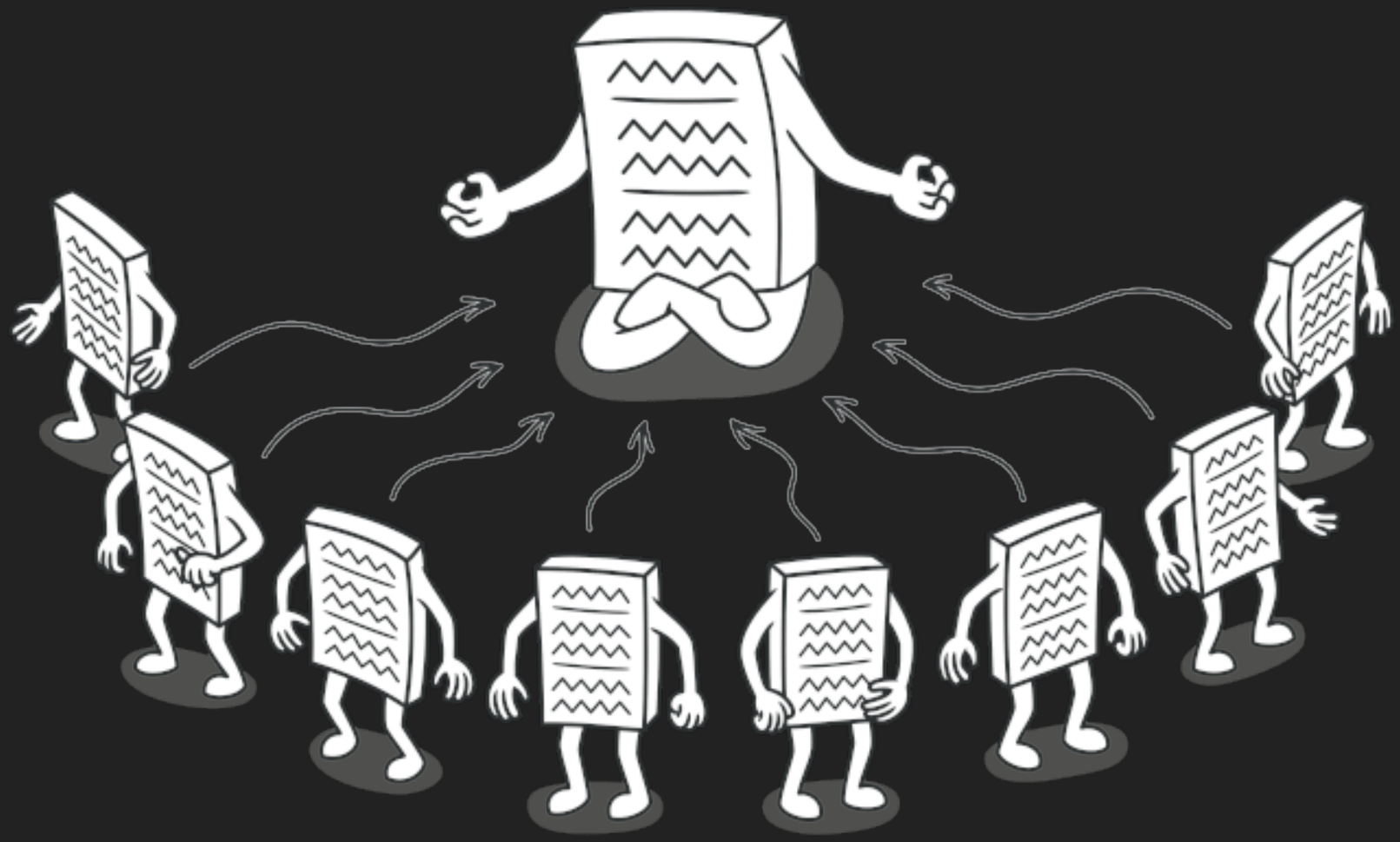
PATRONES DE DISEÑO

NICOLÁS BENÍTEZ
NABENITEZ@UC.CL

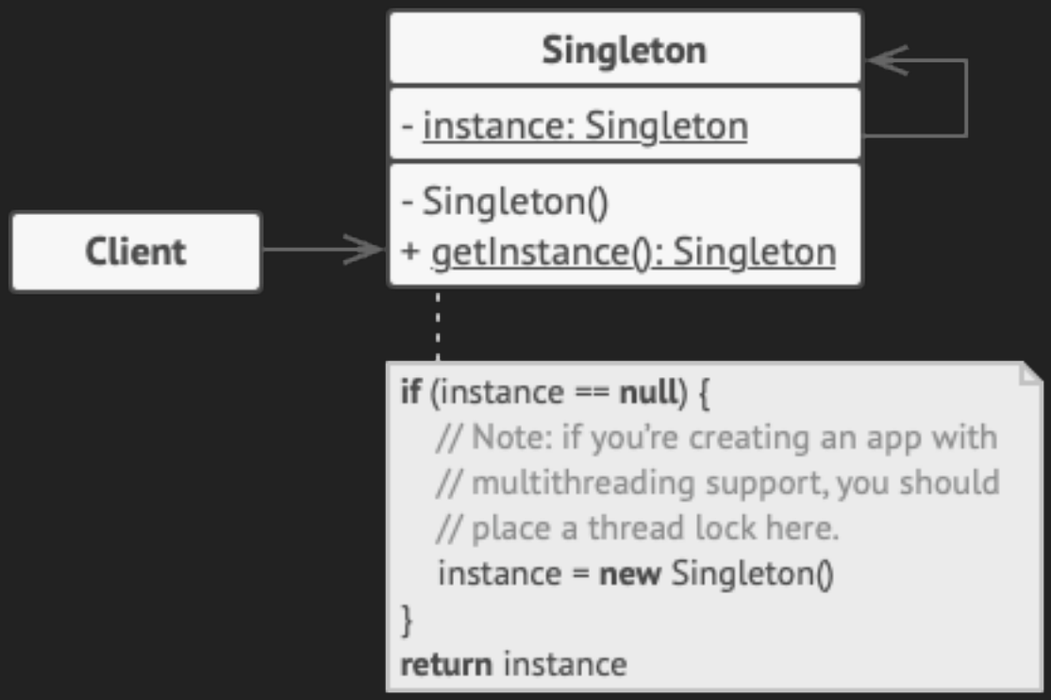
PATRONES CREACIONALES

SINGLETON

- Solo puede existir una instancia de una clase en particular.

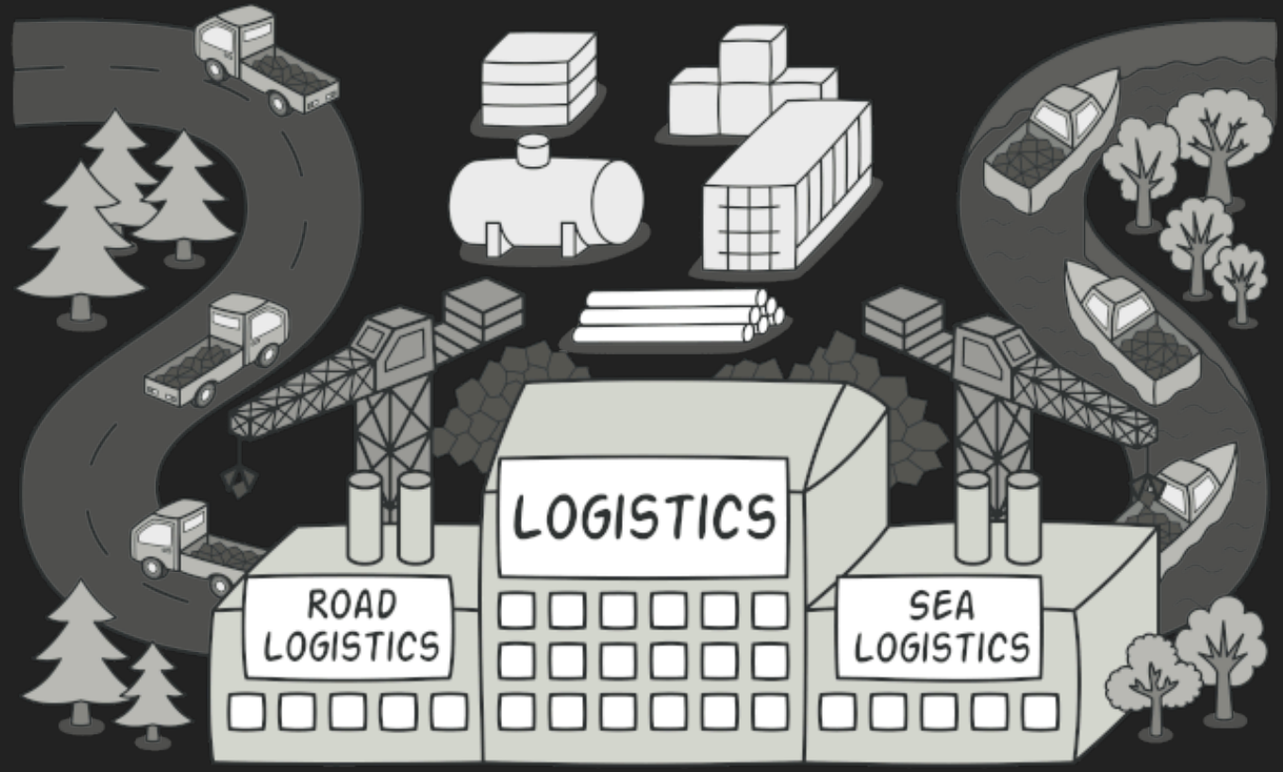


SINGLETON DIAGRAMA UML Y EJEMPLO

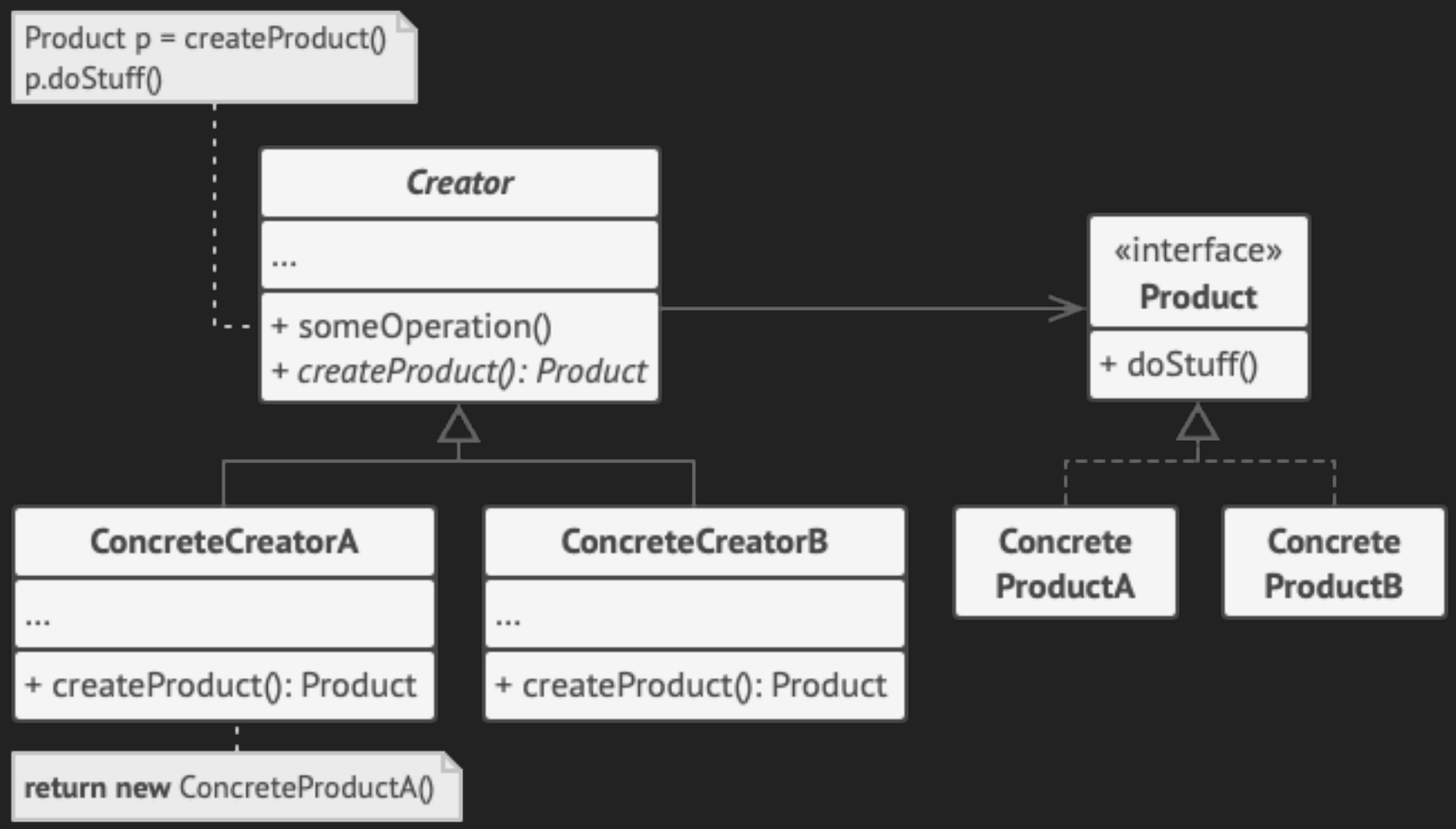


FACTORY

- Permite crear una serie de objetos, indicando el tipo y la cantidad

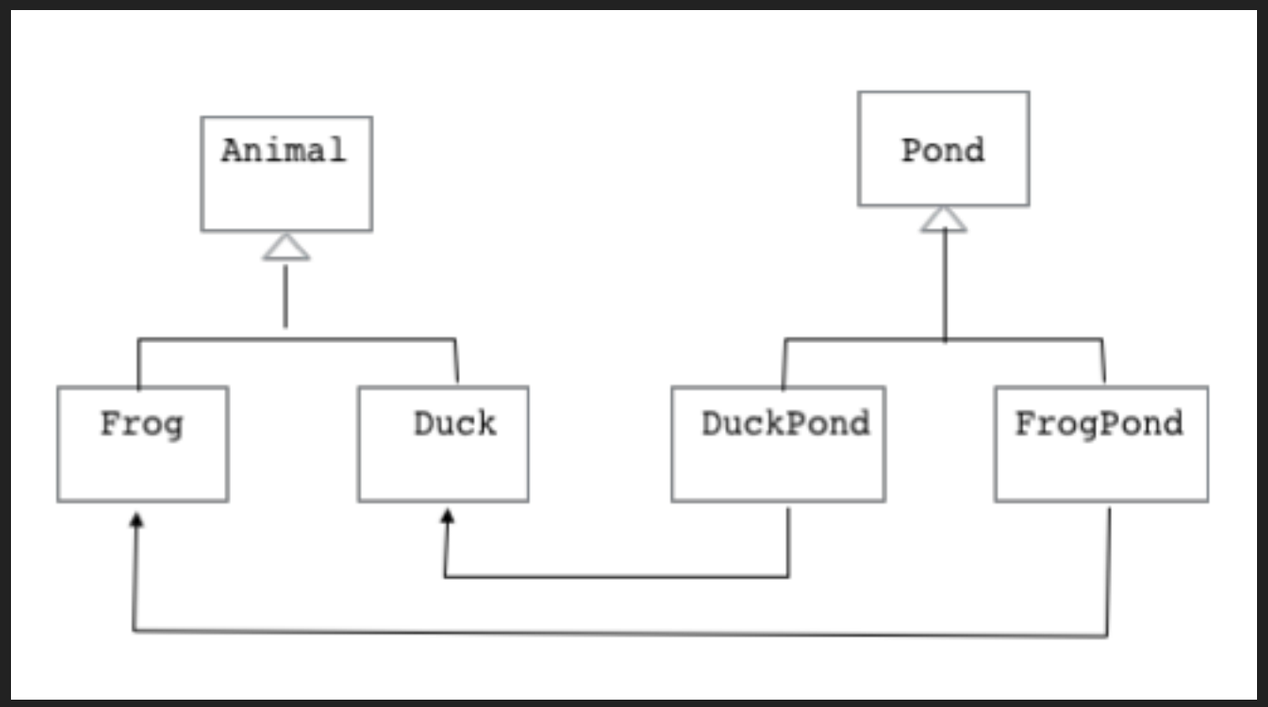


FACTORY DIAGRAMA UML



EJEMPLO: FACTORY

- Laguna de patos.

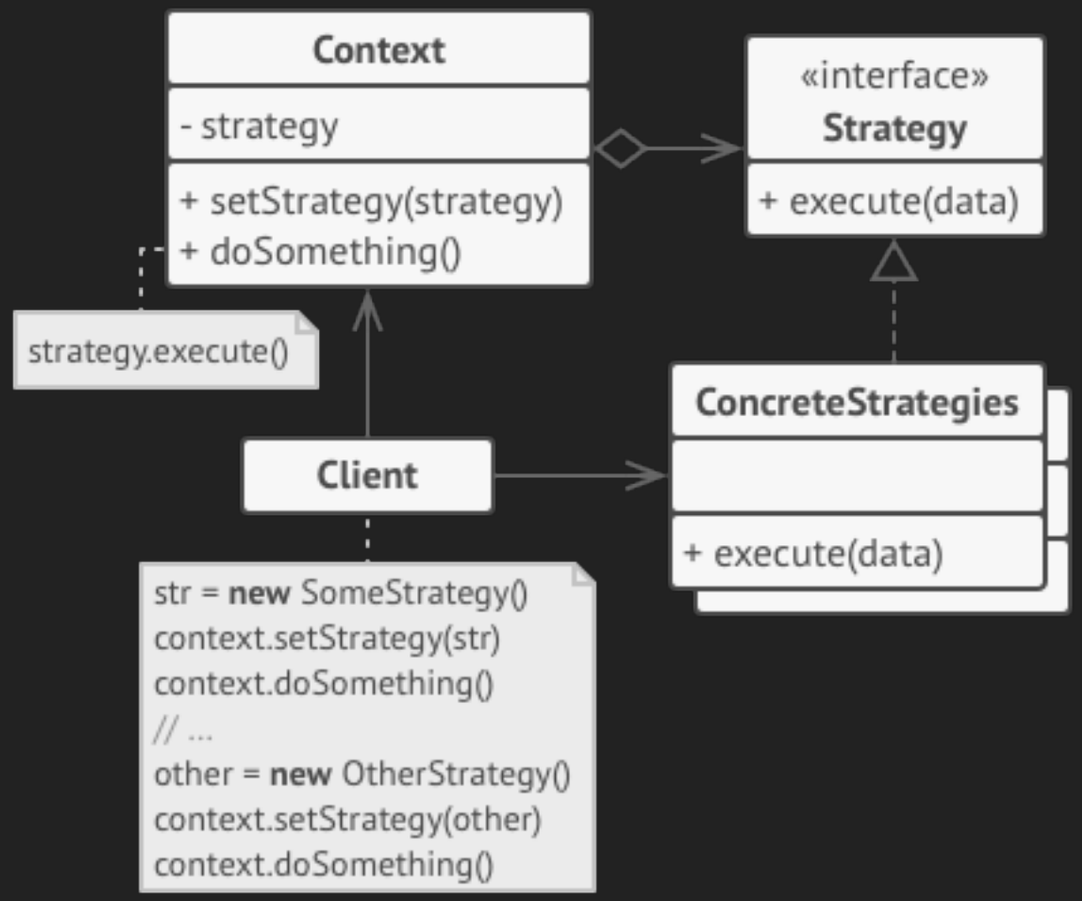


PATRONES DE COMPORTAMIENTO

STRATEGY

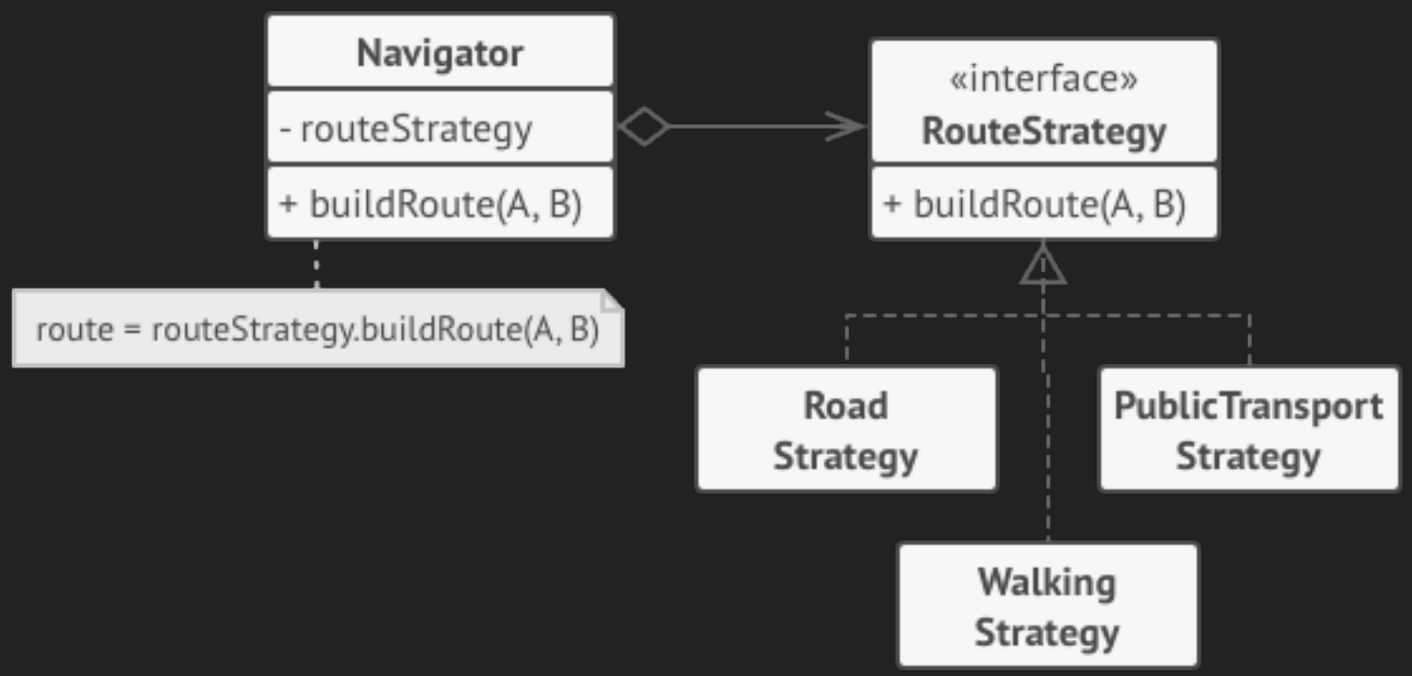
- Permite definir una familia de algoritmos, poniéndolos en clases separadas y haciendo sus objetos intercambiables.

DIAGRAMA STRATEGY



EJEMPLO: GPS

- Implementar un GPS que permita generar la ruta de diferentes formas, en particular para: bicicleta o caminar. Utilizar el diagrama a continuación como referencia.



OBSERVER

- Permite definir un mecanismo de "suscripción" para notificar a distintos objetos (observadores) acerca de eventos que le ocurren a un objeto en particular (observado).

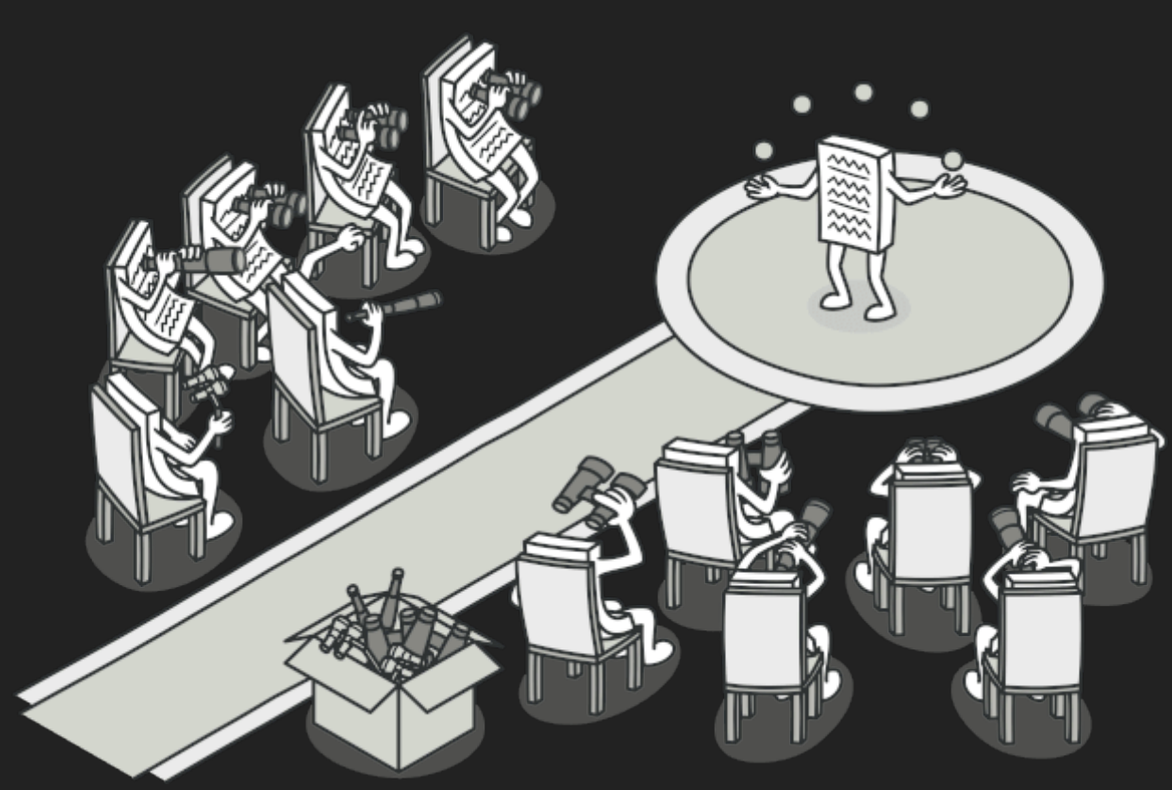
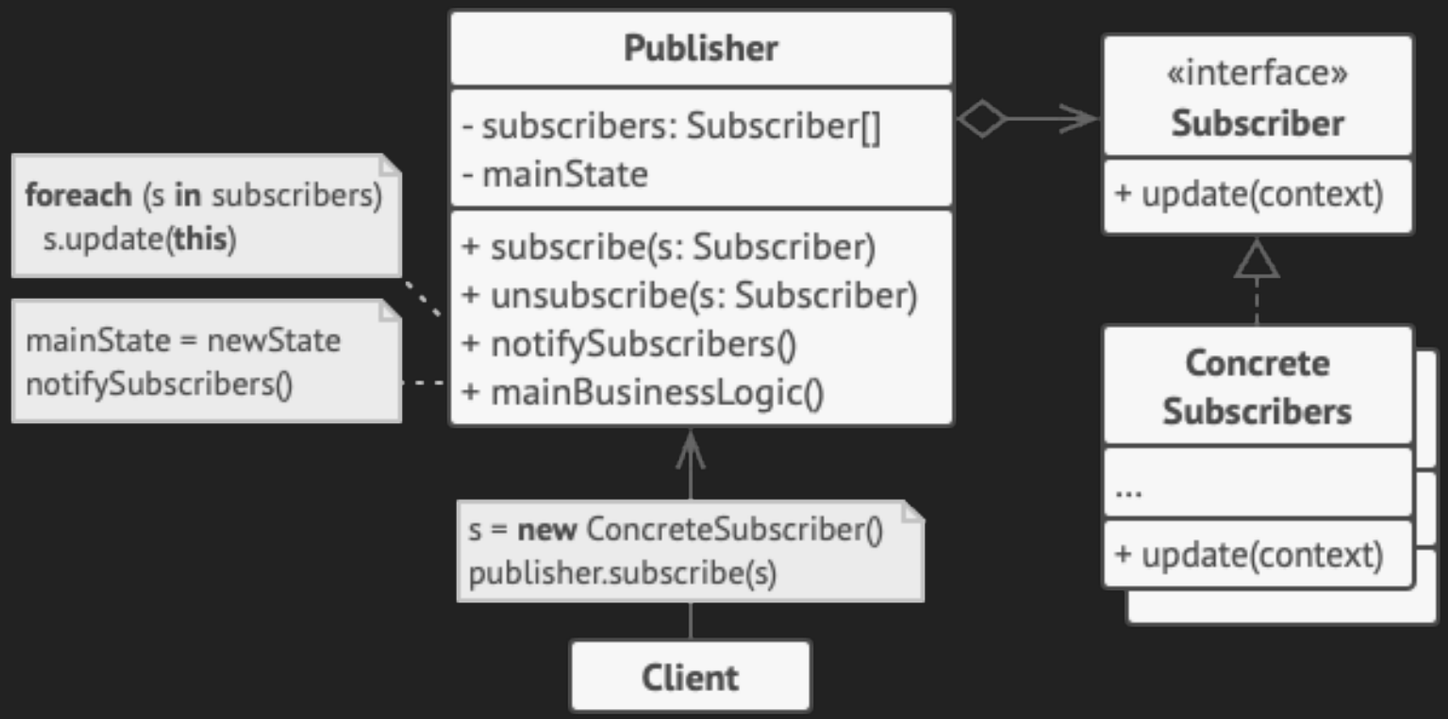


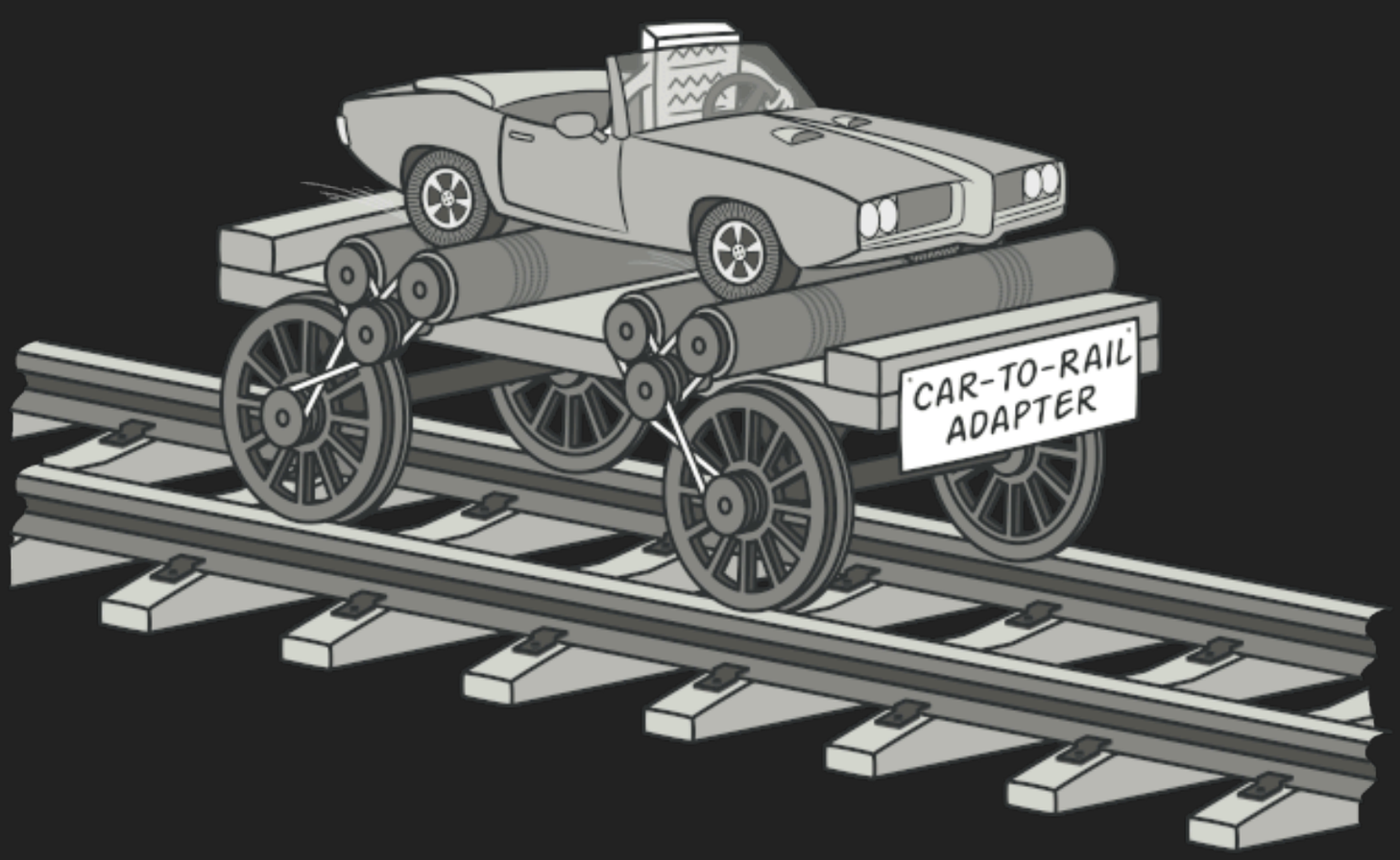
DIAGRAMA OBSERVER Y EJEMPLO



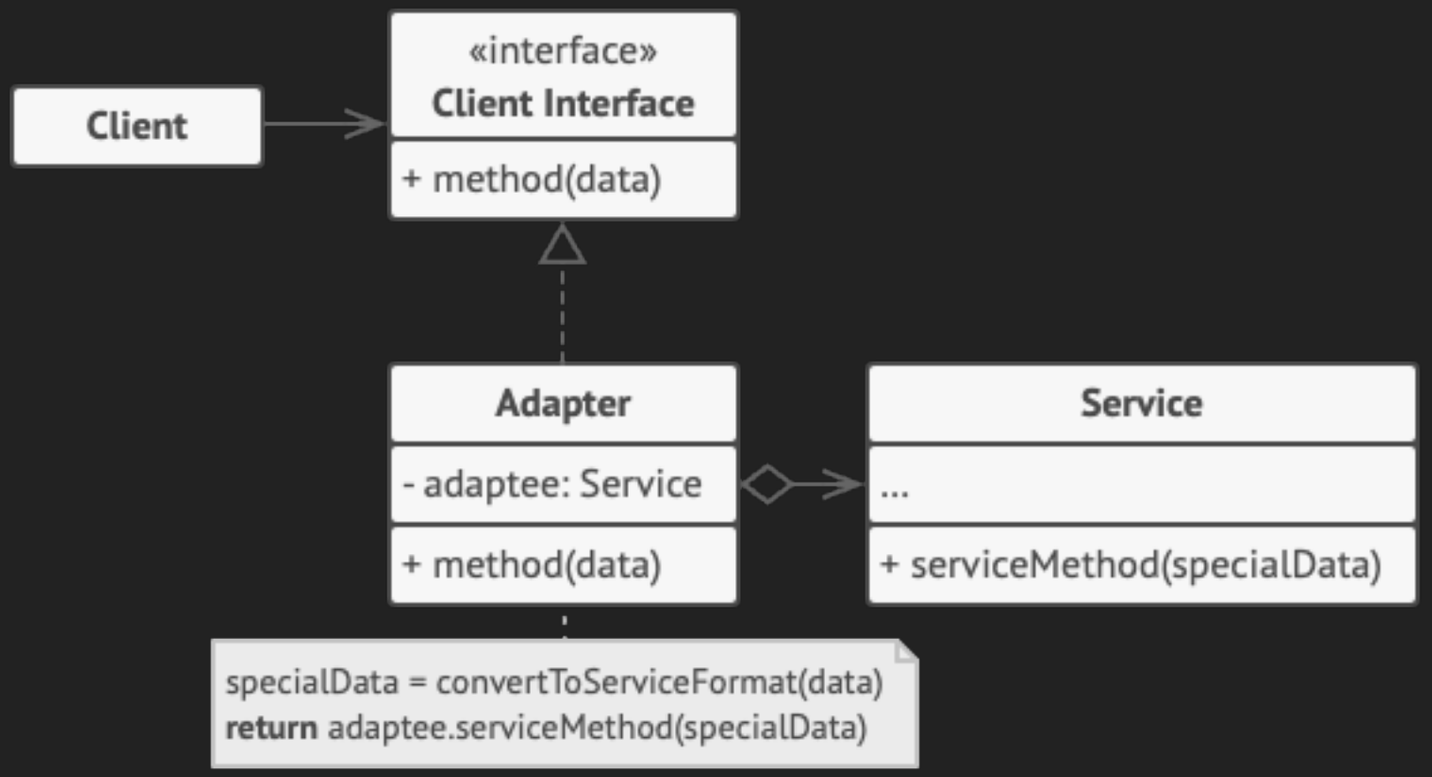
PATRONES ESTRUCTURALES

ADAPTER

- Como su nombre lo dice, su función es adaptar, el problema que resuelve este patrón es hacer trabajar en conjunto objetos que tienen interfaces diferentes.



ADAPTER DIAGRAMA UML

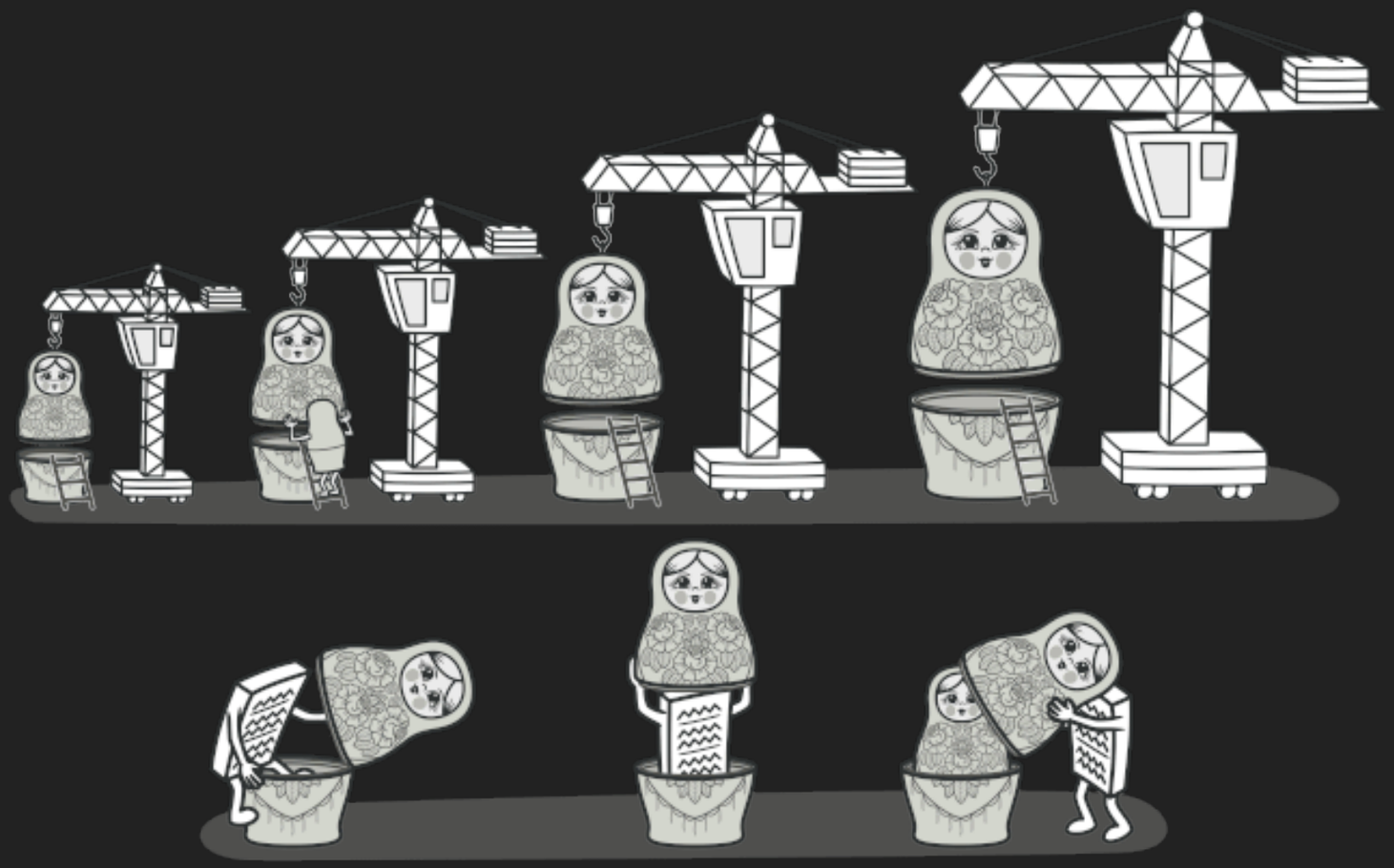


EJEMPLO: ADAPTER

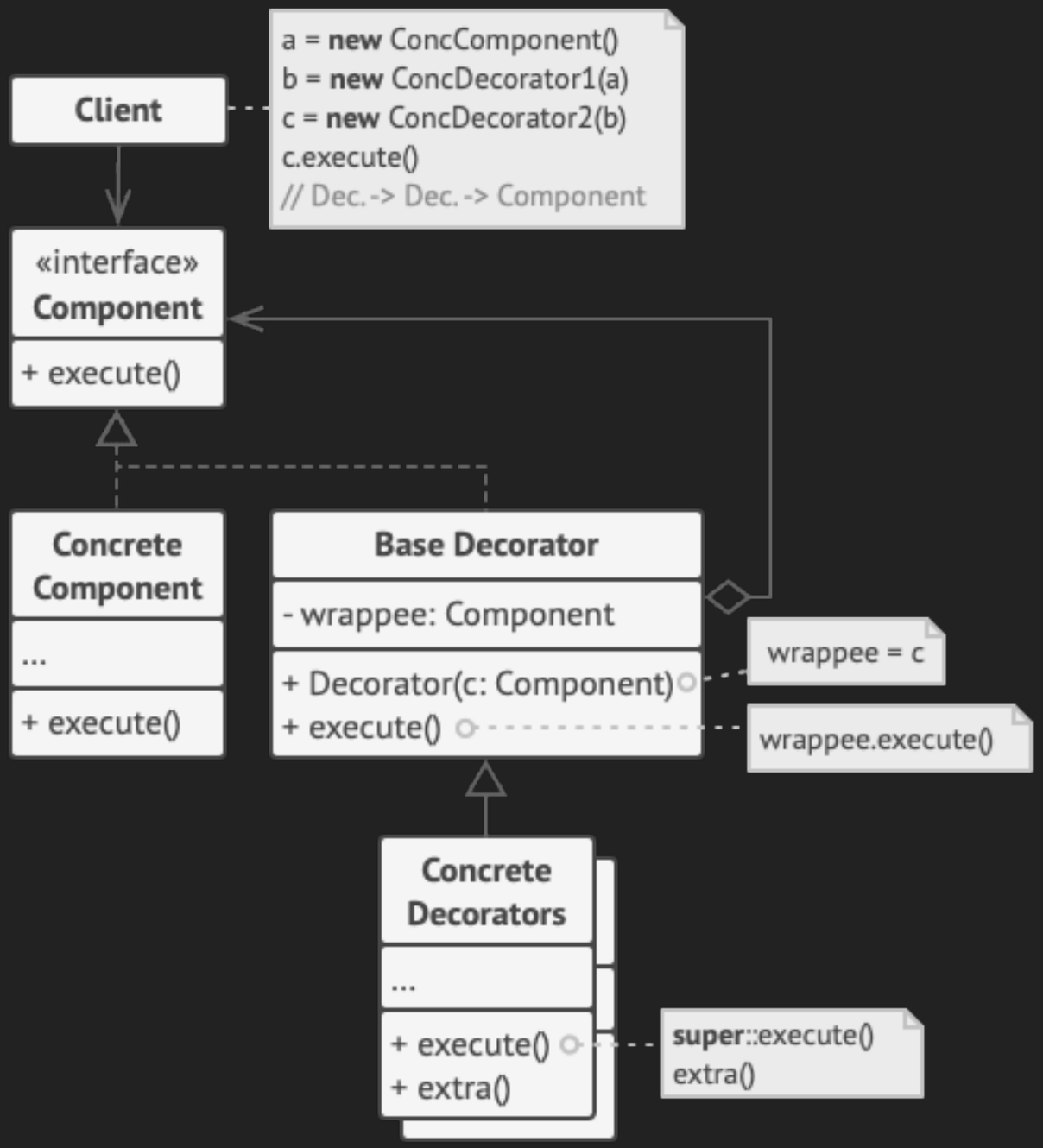
- Se te entrega una función *client_code*, que toma una instancia de *Target* y obtiene el valor de un request, la función sólo es válida para clases que implementan el método *request*. Además, una clase Adaptee que contiene un *specific_request*, debes obtener la información de este con la función client_code.
  No debes modificar nada de la clase Adaptee, ni de la función client_code.
- *specific_request* tiene un *string* que viene invertido, también debes manejar este problema.

DECORATOR

- Permite añadir comportamientos a objetos, poniendo estos en un objeto *wrapper* especial que contiene dichos comportamientos.
- *Wrapper* es un objeto que puede relacionarse con otro y modificar su comportamiento.



DECORATOR DIAGRAMA UML



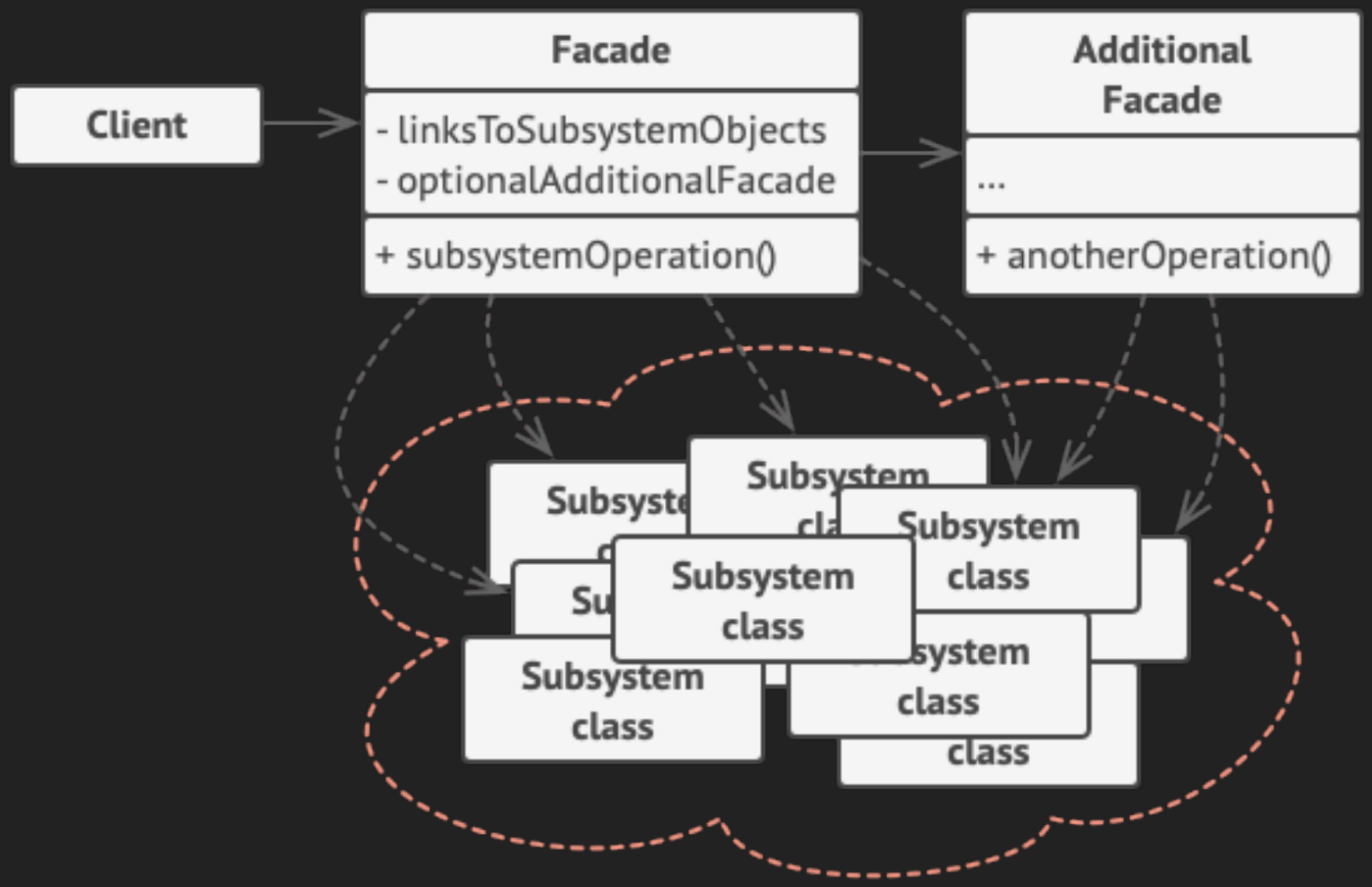
EJEMPLO: DECORATOR

- Tienes una clase MatchA, esta permite saber que A le dio like a una persona B. Sin embargo, necesitas saber si A y B hicieron Match. Es necesario saber si B le dio like a A. Si esto ocurre A y B hicieron match. Debes obtener este comportamiento sin modificar MatchA, definiendo la lógica para saber si B le dio like a A y si A y B hicieron match, a través de decoradores. En particular a través de MatchB y MatchAB.

FACADE

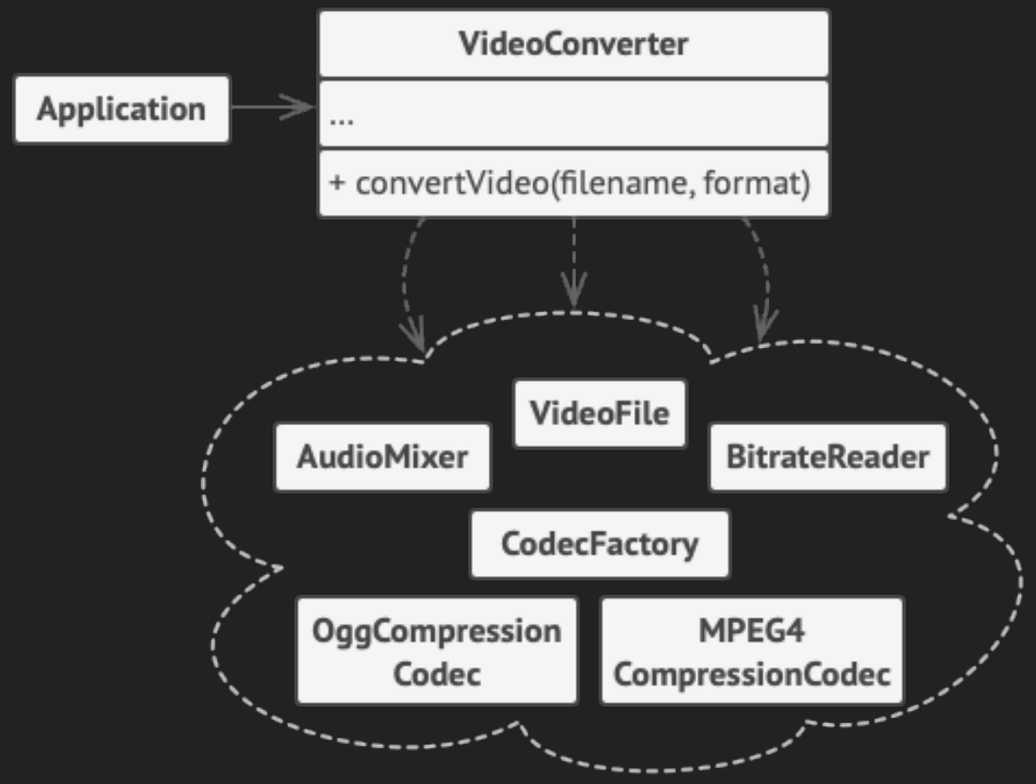
- Provee una interfaz amigable para el cliente, de esta forma se puede acceder a comportamientos complejos de manera simplificada.

FACADE DIAGRAMA UML



EJEMPLO: FACADE

- Se entregan una serie de clases que te ayudan a convertir un video a un formato específico. Existe una aplicación que debe acceder a estas funcionalidades, por simplicidad necesita acceder a esta conversión de forma directa, ya que espera transformar una gran cantidad de videos, por lo tanto, debes solucionar este problema. A continuación, se muestra un esquema que puede ser de ayuda.



REFERENCIAS

- Refactoring Guru. Disponible en: refactoring.guru
- Capítulo 7 - Profesor José Benedetto