POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

13. *Facade*

Luis Fernández Muñoz
https://www.linkedin.com/in/luisfernandezmunyoz
setillofm@gmail.com

INDICE

1. Problema

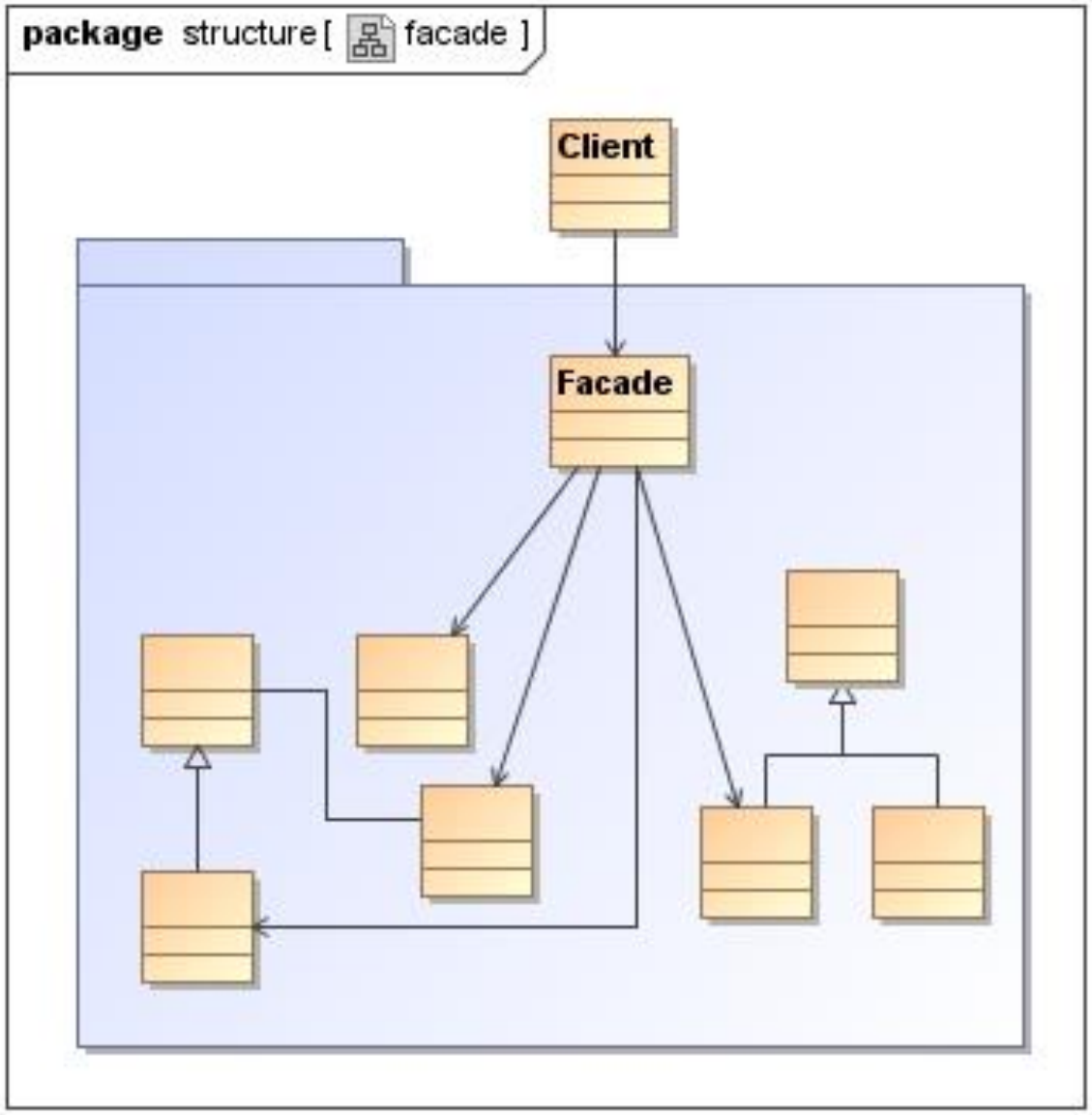
- *Una obra de reforma de un domicilio conlleva que el propietario realice diversas tareas (p.e. solicite licencias, crétditos, …) y, en particular, contrate a diversos especialistas (p.e. albañil, pintor, electricista, decorador, …) y coordine a todos (p.e. tiempos, retrasos, materiales, pre-condiciones, …). Lo cual es muy complejo y tiende a crecer con nuevos "materiales" (p.e. expertos en domótica, …)*
- *Una alternativa más sencilla y que no tiende a crecer es que el propietario se centre en sus tareas y contrate a un contratista para no interactuar con el resto de especialistas.*

- ***Proveer de una interfaz unificada a un conjunto de interfaces de un subsistema definiendo una interfaz de más alto nivel que hace más fácil de usar el subsistema.***

1. Problema

- Los subsistemas a menudo obtienen complejidad según evolucionan. La mayoría de los patrones, cuando se aplican, provocan más clases más pequeñas. Esto hace del subsistema más reusable y fácil de personalizar pero también llega a ser más duro de usar por los clientes que no necesiten personalizar.
- Hay muchas dependencias entre los clientes y las implementaciones de las clases de una abstracción (subsistema).
- Se quiere organizar por capas los subsistemas

2. Solución



2. Solución

- Proveer una vista simple por defecto que es suficiente para la mayoría de los clientes. Solo los clientes que necesiten más particularizaciones necesitarán mirar tras la fachada.
- Desacopla el subsistema de los clientes y otros subsistemas, además de que promociona la independencia del subsistema y la portabilidad
- Permite definir un punto de entrada para cada nivel de subsistema. Si los subsistemas son dependientes, entonces se puede simplificar las dependencias entre ellos haciéndolos comunicarse con los otros solamente a través de sus fachadas

3. Consecuencias

- Promueve débil acoplamiento entre el subsistema y sus clientes
  - Blinda a los clientes de componentes del subsistema, reduciendo así el número de objetos que tratan con los clientes y hacer que el subsistema más fácil de usar.
- No impide que las aplicaciones utilicen clases de subsistema si necesitan. De este modo se puede elegir entre la facilidad de uso y la generalidad
- Ayuda a la organización por capas de un sistema
- Simplifica el traslado de los sistemas a otras plataformas
- La reducción de las dependencias de compilación es vital en grandes sistemas de software.

4. Relaciones

- ***Singleton*** para controlar una única *Facade*

5. Comparativa

- ***Facade** vs **Mediator***
  - Son similares porque abstraen funcionalidad a partir de unas clases existentes. Sin embargo:
    - *Facade* simplemente abstrae una interfaz para los objetos del subsistema, haciéndolos más fácil de usar; no define nueva funcionalidad, y las clases del subsistema no saben de su existencia. Es decir, los objetos *Facade* hacen peticiones a las clases del subsistema pero no a la inversa, su protocolo es unidireccional
    - *Mediator* abstrae cualquier comunicación entre objetos similares, a menudo centralizando la funcionalidad que no pertenece a ninguno de ellos. Los colegas de *Mediator* sólo se preocupan de comunicarse con él y no entre ellos directamente. Permite un comportamiento cooperativo que no es proporcionado por los objetos colegas y el protocolo es multidireccional

5. Comparativa

- *Facade vs Abstract Factory*
  - Ver *Abstract Factory vs Facade*