|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | Diseño | | |
| AREA | Software | REGIMEN | Semestral |

**Tema: Patrones de diseño**

[**Facade**](https://migranitodejava.blogspot.com/2011/06/facade.html)

Busca simplificar el sistema, desde el punto de vista del cliente, proporcionando una interfaz unificada para un conjunto de subsistemas, definiendo una interfaz de nivel más alto. Esto hace que el sistema sea más fácil de usar.  
Este patrón busca reducir al mínimo la comunicación y dependencias entre subsistemas. Para ello, utilizaremos una fachada, simplificando la complejidad al cliente. El cliente debería acceder a un subsistema a través del Facade. De esta manera, se estructura un entorno de programación más sencillo, al menos desde el punto de vista del cliente (por ello se llama "fachada").  
Se debe utilizar cuando:

* Se quiera proporcionar una interfaz sencilla para un subsistema complejo.
* Se quiera desacoplar un subsistema de sus clientes y de otros subsistemas, haciéndolo más independiente y portable.
* Se quiera dividir los sistemas en niveles: las fachadas serían el punto de entrada a cada nivel. Facade puede ser utilizado a nivel aplicación.

**Diagrama UML**

A diagram of a system

Description automatically generated

Facade:  conoce cuales clases del subsistema son responsables de una petición.  
Delega las peticiones de los clientes en los objetos del subsistema.  
Subsistema: manejar el trabajo asignado por el objeto Facade. No tienen ningún conocimiento del Facade (no guardan referencia de éste).  
Los clientes se comunican con el subsistema a través de la facade, que reenvía las peticiones a los objetos del subsistema apropiados y puede realizar también algún trabajo de traducción. Los clientes que usan la facade no necesitan acceder directamente a los objetos del sistema.  
**Ejemplo**  
Imaginemos que estamos, con un equipo de desarrollo, realizando el software para una inmobiliaria. Obviamente una inmobiliaria realiza muchos trabajos diferentes, como el cobro de alquiler, muestra de inmuebles, administración de consorcios, contratos de ventas, contratos de alquiler, etc.  
Por una cuestión de seguir el paradigma de programación orientada a objetos, es probable que no se realice todo a una misma clase, sino que se dividen las responsabilidades en diferentes clases.  
Imaginemos que en el soft de la inmobiliaria tenemos diversos tipos de Personas, todas con sus atributos y métodos correspondientes. Aca pondré sólo un resumen de ellas, ya que no tiene sentido ahondar en demasiados detalles de implemetación.

A close-up of words

Description automatically generated

Lo mismo con los métodos principales de las diversas clases que tiene el sistema:A screenshot of a computer code

Description automatically generated

Ahora haremos una clase llamada Inmobiliaria que será nuestro Facade:A screenshot of a computer program

Description automatically generated

Bien, veamos ahora 2 tipos de clientes. El primero no llamará al Facade y el segundo si lo utilizará. Veremos como el primero esta obligado a conocer muchos detalles de los subsistemas y el segundo no.

A computer screen shot of a code

Description automatically generated

**Consecuencias**

* Oculta a los clientes de la complejidad del subsistema y lo hace fácil de usar.
* Favorece un acoplamiento débil entre el subsistema y sus clientes, consiguiendo que los cambios de las clases del sistema sean transparentes a los clientes.
* Facilita la división en capas y reduce dependencias de compilación.
* No se impide el acceso a las clases del sistema.

Temas a tener en cuenta.  
Lo más importante de todo es que este patrón se debe aplicar en las clases más representativas y no en las específicas. De no ser así, posiblemente no se tenga el nivel alto deseado.  
Por aplicación, es ideal construir no demasiados objetos Facade. Sólo algunos representativos que contengan la mayoría de las operaciones básicas de un sistema.