POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

22. *Proxy*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

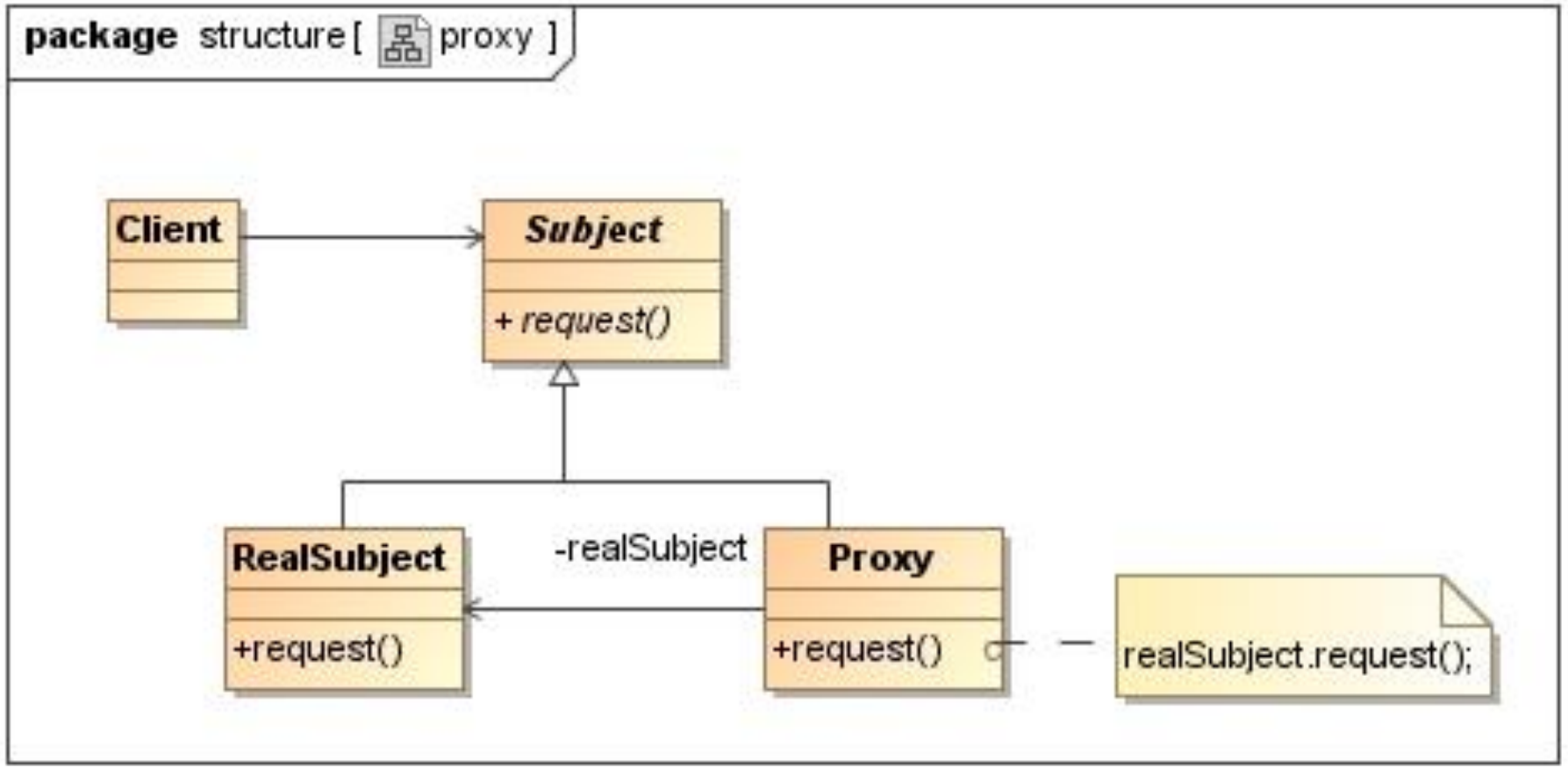
- *Cuando un jefe necesita abandonar la oficina se desatienden diversos aspectos en la toma de decisiones (p.e. reacción ante cierta contingencia, concesión de vacaciones, …) que requieren los empleados para su trabajo diario. Esto provoca una dependencia innecesaria en la localización del jefe para que todo siga su curso.*
- *Una solución alternativa que independiza la localización del jefe es nombrar a un representante en su ausencia para que centralice la atención de las demandas de los empleados, las notifique al jefe (p.e. por teléfono o correo o …) y responda con las decisiones de éste sin tener que delegar en nadie.*

- ***Proporcionar un sustituto o representante de otro objeto para controlar el acceso al mismo.***

1. Problema

- Es aplicable cada vez que hay necesidad de una referencia a un objeto más versátil o sofisticada que un simple puntero:
  - Una razón para controlar el acceso a un objeto es retrasar todo el coste de su creación e inicialización hasta que sea realmente necesario usarlo. Un *proxy* virtual crea objetos costosos por encargo
  - Un *proxy* remoto proporciona un representante local de un objeto situado en otro espacio de direcciones
  - Un *proxy* de protección que controla el acceso al objeto original. Los *proyies* de protección son útiles cuando los objetos deberían tener diferentes permisos de acceso
  - Un referencia inteligente es un sustituto de un simple puntero que lleva a cabo operaciones adicionales cuando se accede a un objeto: contar el número de referencias, …

2. Solución


package structure [proxy]
Client
Subject
+ request()
RealSubject
+request()
-realSubject
Proxy
+request()
realSubject.request();

2. Solución

- C++ admite la sobrecarga del operador de acceso a miembros. Sobrecargar este operador permite realizar tareas adicionales cada vez que se des-referencia un objeto, lo que puede resultar útil para implementar algunos tipos de *proxies*; el *proxy* se comporta igual que un puntero
- Si una clase *Proxy* puede tratar con su sujeto sólo a través de una interfaz abstracta, entonces no hay necesidad de hacer una clase *Proxy* para cada clase de SujetoReal; el proxy puede tratar de manera uniforme a todas las clases SujetoReal. Pero si los objetos *Proxy* van a crear instancias de SujetoReal, entonces tienen que conocer la clase concreta.

3. Consecuencias

- Introduce un nivel de indirección al acceder a un objeto. Esta indirección adicional tiene muchos posible usos, dependiendo del tipo de *proxy*:
  - ocultar el hecho de que un objeto reside en un espacio de direcciones diferente;
  - optimizaciones tales como crear un objeto por encargo; y
  - realizar tareas de mantenimiento adicionales cuando se accede a un objeto.

4. Comparativa

- ***Proxy*** vs ***Adapter***.
  - Ver *Adapter* vs *Proxy*
- ***Proxy*** vs ***Decorator***.
  - Ver *Decorator* vs *Proxy*