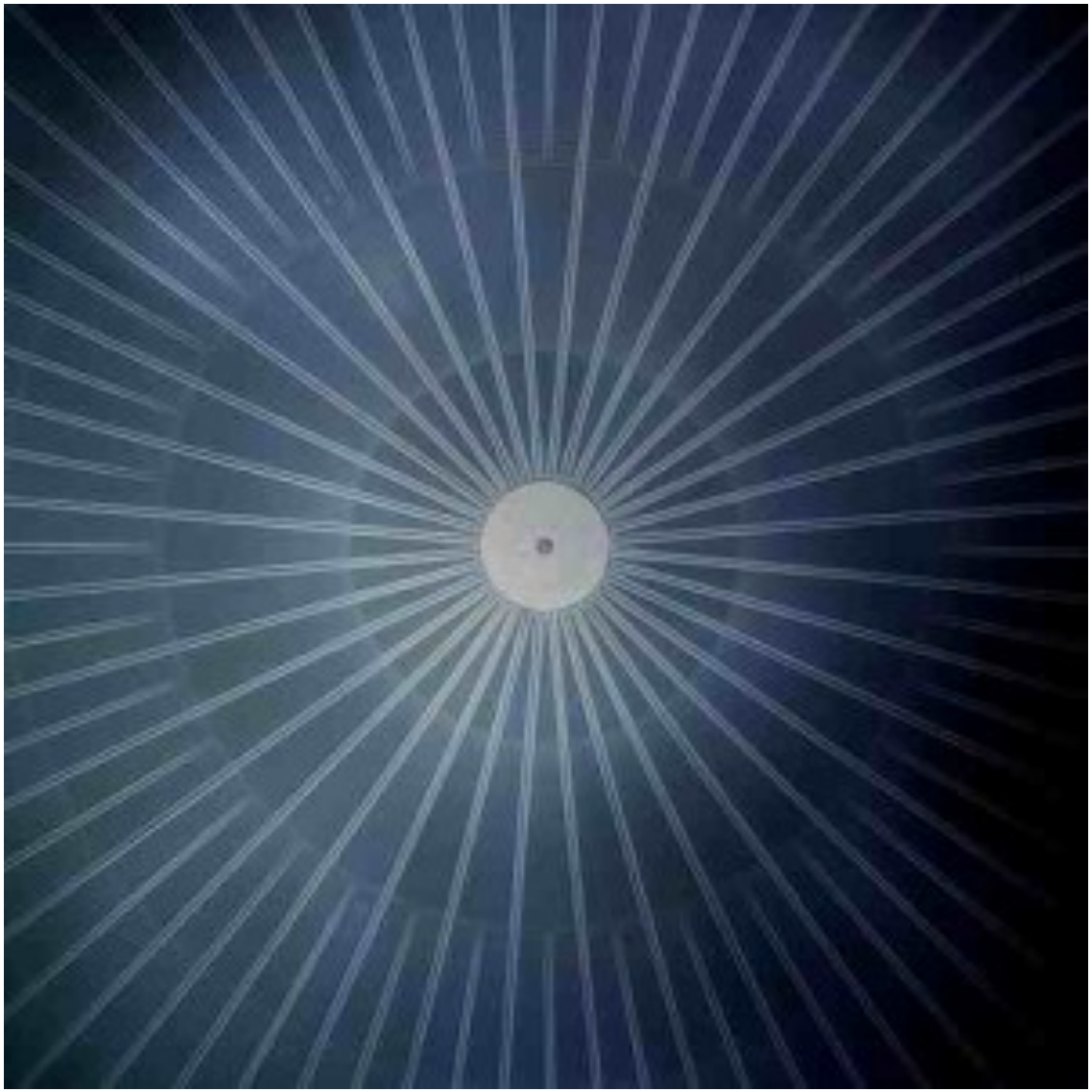
2do nuevo patrón



Ejemplo

De Carolyn Nelson <coney@reczook.loan>

Responder | Reenviar | Archivar | Basura | Borrar | Más

Asunto ***SPAM-SOL*** Regrows Lost Hair After 19 Days

3:17 p.

A Yo

Para proteger su privacidad, Thunderbird ha bloqueado el contenido remoto en este mensaje.

Opciones

SHARK TANK - GET YOUR HAIR BACK

2 Sprays On Your Scalp And Its Back In 4 Days

Voted Best NEW Product of 2017 by the Sharks Look at how easy and eff

Carga bajo demanda – Fuerzas del problema

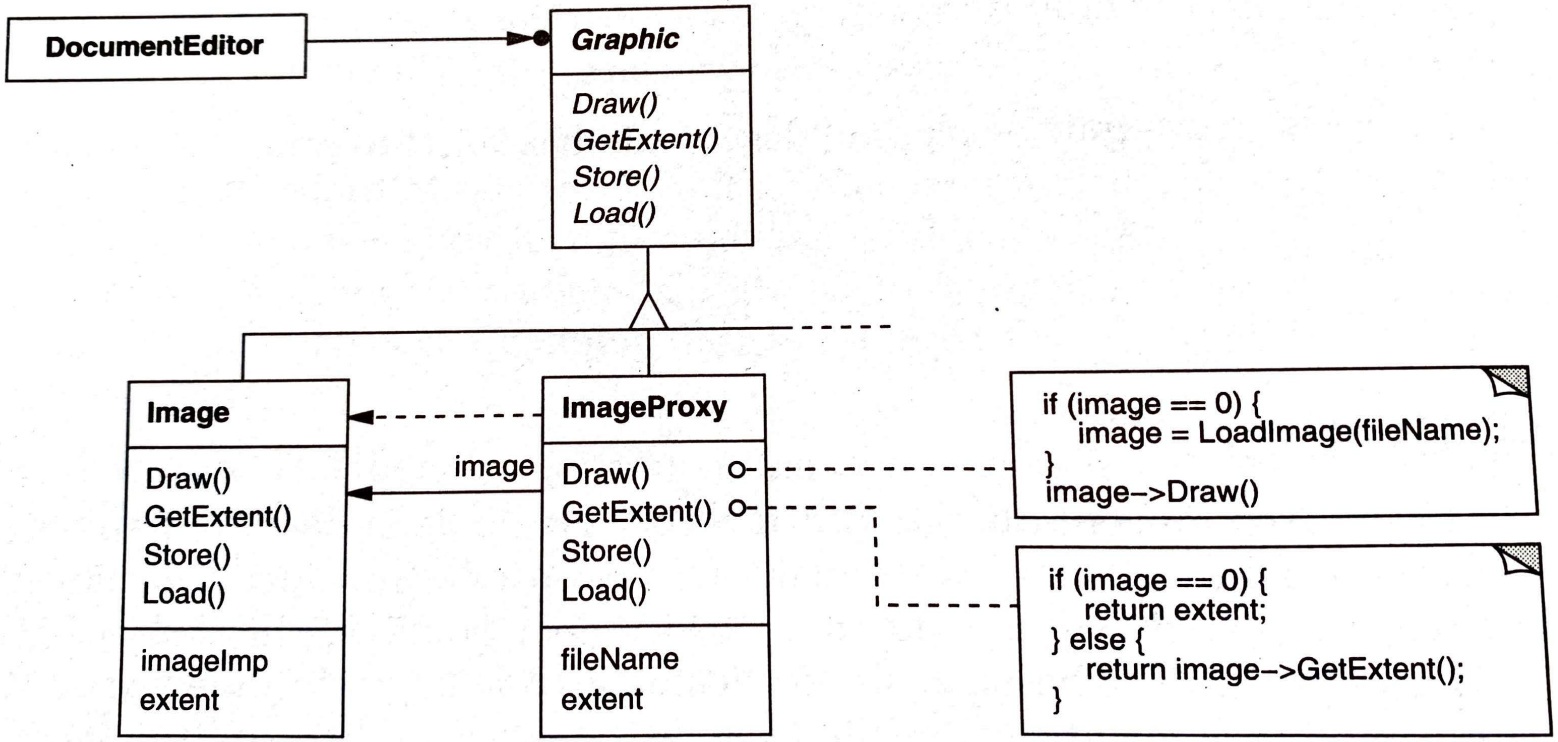
- En muchos casos un email puede tener muchas imágenes, siendo estas pesadas y lentas de cargar
- No queremos que la apertura de un email sea lenta.
- En algunos casos las imágenes ni siquiera serán vistas.
- Queremos evitar el costo de leer la imagen hasta tanto sea necesario mostrarla
- Igualmente necesitamos un « representante » de la imagen, de manera de darle al cliente un objeto que se vea y actúe como el cliente espera

Carga bajo demanda: Solución

- La idea es crear una imagen “falsa”, un impostor que
  - Debe responder a los mensajes de la imagen verdadera (mantiene el protocolo).
  - Sabe responder a algunos mensajes (tamaño de la imagen)
  - Cuando sea necesario mostrarla en pantalla, debe ir a buscar la imagen original al servidor, leerla y mostrarla.

Solución

- Cargar las imágenes bajo demanda, utilizando un objeto *proxy*. El proxy se comporta como una imagen normal y es el responsable de cargar la imagen bajo demanda



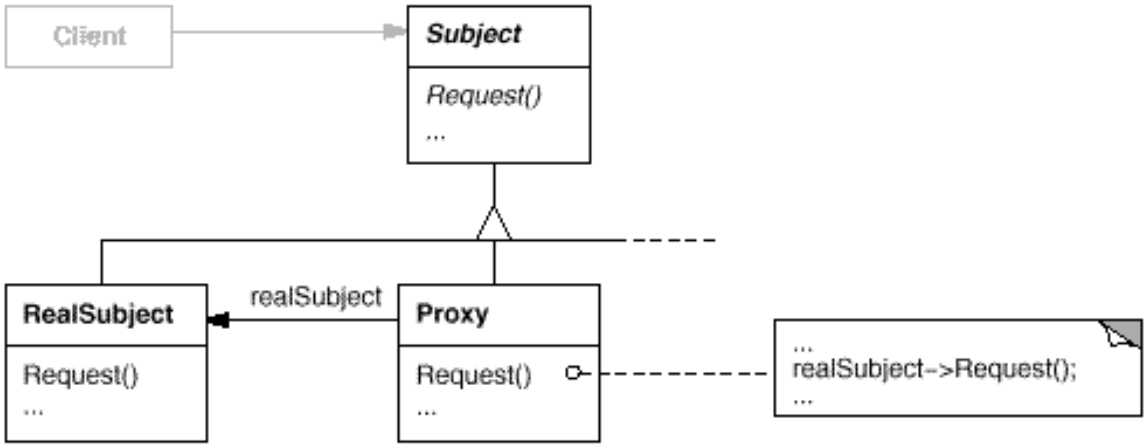
Patrón Proxy

- **Propósito:** proporcionar un intermediario de un objeto para controlar su acceso.

- **Aplicabilidad**: cuando se necesita una referencia a un objeto más flexible o sofisticada

Patrón *Proxy*. Solución

- Colocar un objeto intermedio que respete el protocolo del objeto que está reemplazando.
- Algunos mensajes se delegarán en el objeto original. En otros casos puede que el proxy colabore con el objeto original o que reemplace su comportamiento.



Patrón *Proxy*

- **Aplicaciones del proxy:**
  - Demorar la construcción de un objeto hasta que sea realmente necesario (**virtual proxy**).
  - Restringir el acceso a un objeto por seguridad (**protection proxy**).
  - Implementación de objetos distribuídos (**remote proxy**).

Proxy de protección

```
BankAccount
    v.i. balance

>>balance
    ^balance

>>deposit: anAmount
    balance := balance + anAmount

>>withdraw: anAmount
    balance := balance - anAmount
```

```
BankAccountProxy
    v.i. realAccount

>>balance
    self checkAccess ifTrue: [
        ^realAccount balance]

>>deposit: anAmount
    self checkAccess ifTrue: [
      ^realAccount deposit: anAmount]

>>withdraw: anAmount
    self checkAccess ifTrue: [
      ^realAccount withdraw:
                        anAmount]

>>checkAccess ...
```

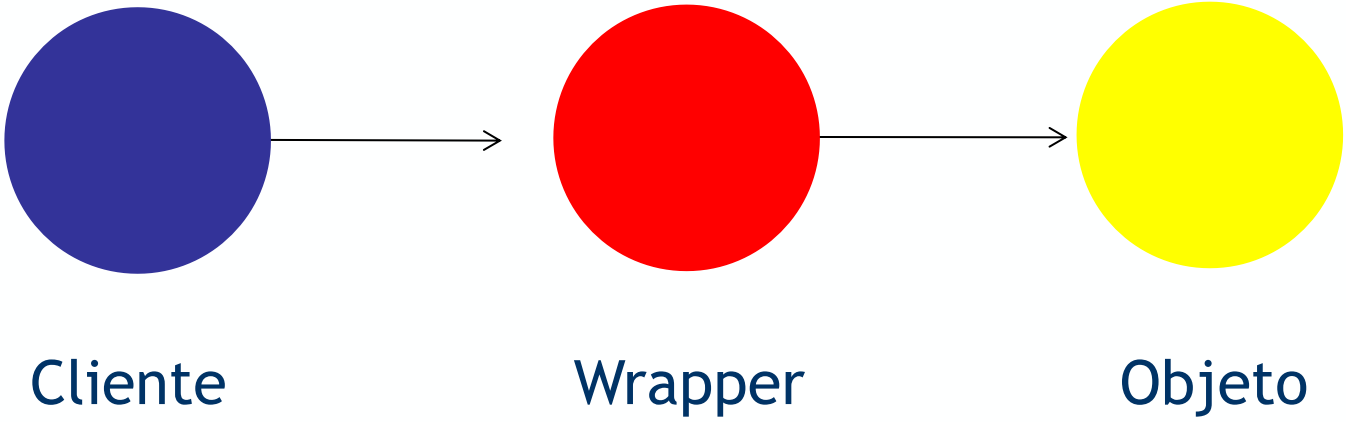
Proxy de acceso remoto

- Para acceder a objetos que se encuentran en otro espacio de memoria, en una arquitectura distribuida
- El proxy empaqueta el request, lo envía a través de la red al objeto real, espera la respuesta, desempaqueta la respuesta y retorna el resultado
- En este contexto el proxy suele utilizarse con otro objeto que se encarga de encontrar la ubicación del objeto real. Este objeto se denomina **Broker**, del patrón de su mismo nombre

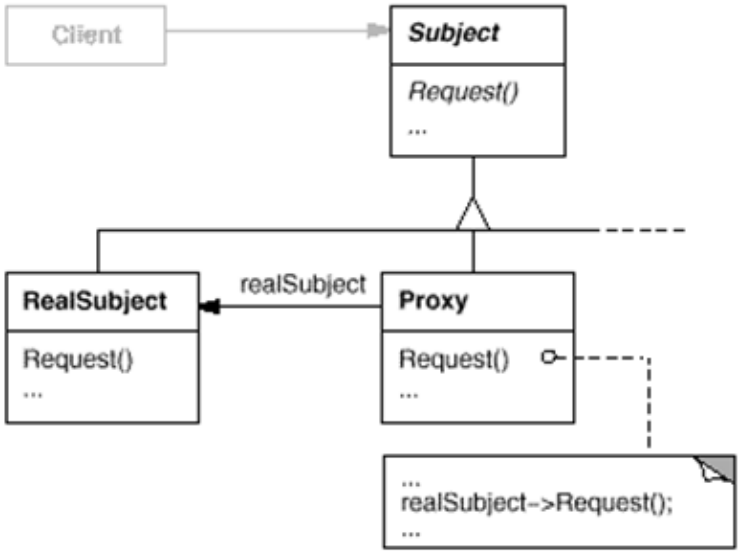
Adapter, Decorator, Proxy

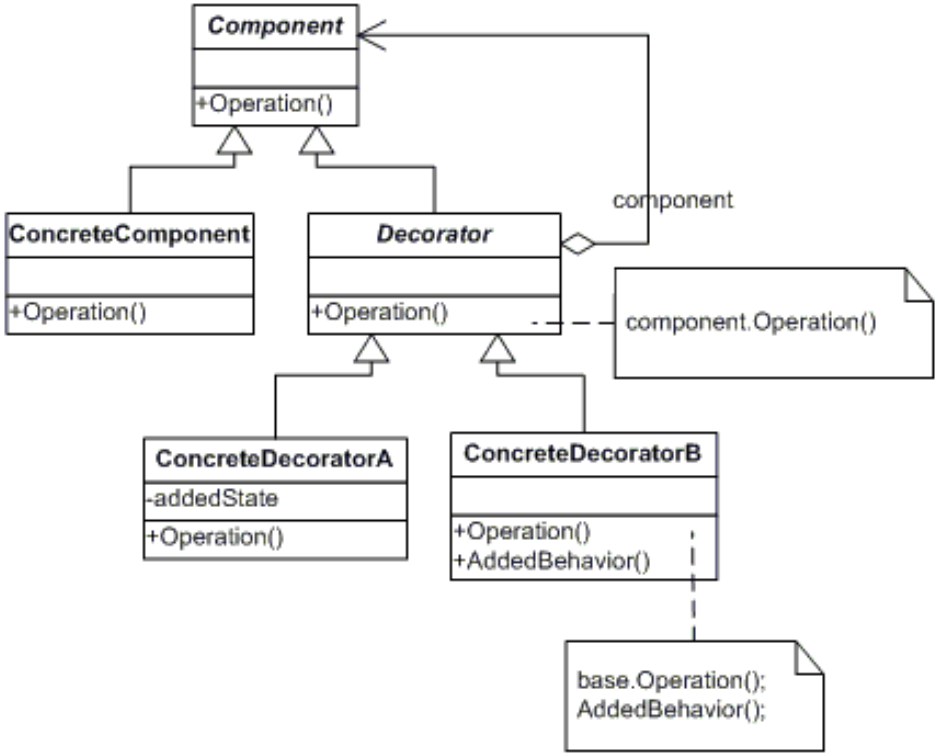
- Todos patrones estructurales
- Todos con diagramas de objetos similares
- Distinto propósito

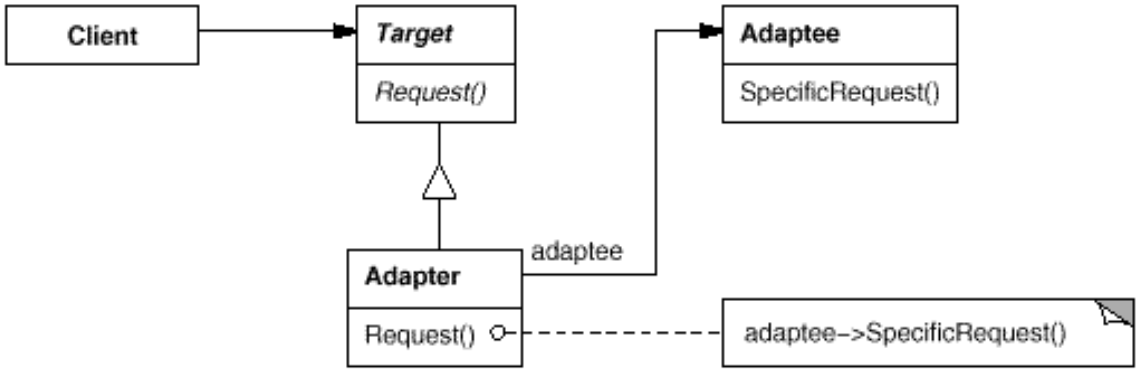
- A todos se los llama también “wrappers”



Proxy vs. Decorator vs. Adapter







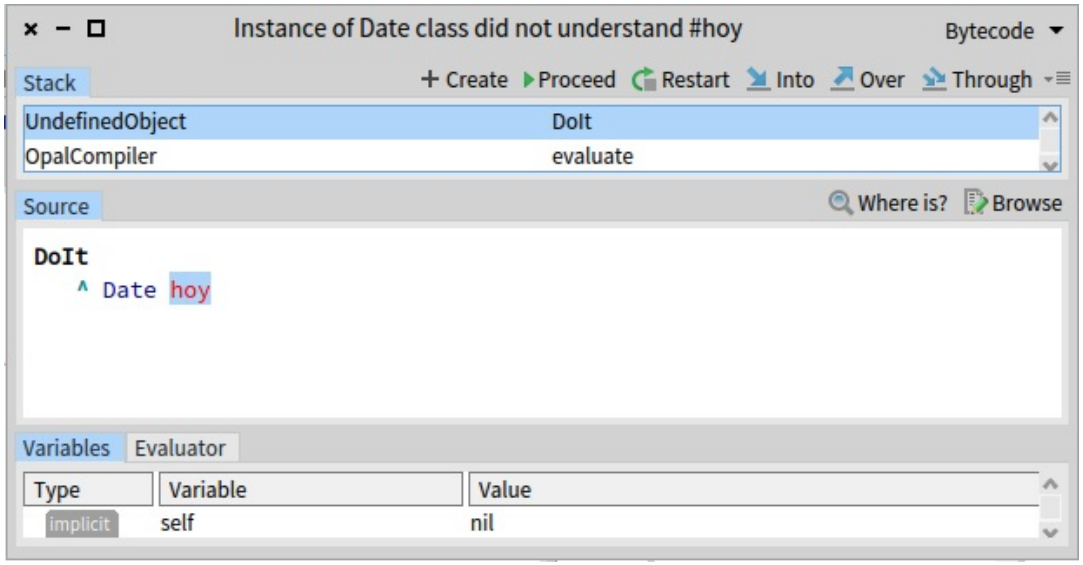
Patrón *Proxy*

- **Implementación:**
  - Redefinir todos los mensajes del objeto real ??
  - Proxy no siempre necesita conocer la clase del objeto real

Implementación de Proxy usando reflexión

- Method look-up
- #doesNotUnderstand: aMessage

- Cuando a un objeto se le envía un mensaje que no implementa, la máquina virtual le envía el mensaje #doesNotUnderstand: al objeto con una “reificación” del mensaje como argumento.
- El mensaje (instancia de Message) contiene al selector y un Array de los argumentos.



Implementación de Proxy usando reflexión

- En el método #doesNotUnderstand: podriamos examinar el contexto en el que ocurrió el error, cambiarlo y continuar la ejecución
- ➔ En vez de reimplementar en Proxy todos los mensajes, solo se define #doesNotUnderstand:
- ➔ Cada envío de mensaje a instancias de Proxy termina en #doesNotUnderstand:, donde el objeto puede manipular el mensaje para por ejemplo, enviárselo al objeto real

#doesNotUnderstand:

```
ImageProxy>>doesNotUnderstand: aMessage
  | image |
  image := CachedImage on:
          (ImageReader fromFile: fileName) image.

  image perform: aMessage selector
          withArguments: aMessage arguments
```

```
BankAccountProxy>>doesNotUnderstand: aMessage
...
```

Entendiendo qué es reflexión

- Un programa reflexivo es aquel que puede razonar sobre si mismo, es decir que puede ***observarse*** y ***cambiarse*** dinámicamente.

- Es aquel que puede observar su propia ejecución (***introspección***) e incluso cambiar la manera en que se ejecuta (***intercesión***).

- Requiere poder expresar y manipular el estado de la ejecución como datos: ***reification***
  - *Reify: to regard (something abstract) as a material or concrete thing*