

DISEÑO Y ARQUITECTURA DE SOFTWARE

Proxy

A series of several parallel white lines of varying lengths, slanted diagonally from the bottom left towards the top right, located on the right side of the slide.

NOMBRE Y CLASIFICACIÓN

Proxy pattern
(representante)



Se clasifica dentro de los patrones de diseño estructurales. Estos hacen más sencilla la fase de diseño identificando maneras simples de llevar a cabo estas relaciones entre clases y objetos.

INTENCIÓN

- ↓ Proporcionar un representante de otro objeto, por distintas razones como pueden ser el acceso, la velocidad o la seguridad, entre otras.

OTROS NOMBRES

↴ Surrogate (sustituto)


A series of several parallel white lines of varying lengths and slopes, extending from the middle right towards the bottom right corner of the slide.

MOTIVACIÓN

Retrasar el coste de crear e inicializar un objeto hasta que es realmente necesario. Por ejemplo, no abrir las imágenes de un documento hasta que no son visibles.

Puede haber ocasiones en que se desee posponer el coste de la creación de un objeto hasta que sea necesario usarlo.

El objeto proxy actúa en lugar del verdadero objeto, y ofrece la misma interfaz, y las solicita en el objeto cuando es necesario.

Several white diagonal lines of varying lengths and thicknesses are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.

APLICACIÓN

El patrón *proxy* se usa cuando se necesita una referencia a un objeto más flexible o sofisticada que un puntero. Dependiendo de la función que se desea realizar con dicha referencia podemos distinguir diferentes tipos de *proxies*:

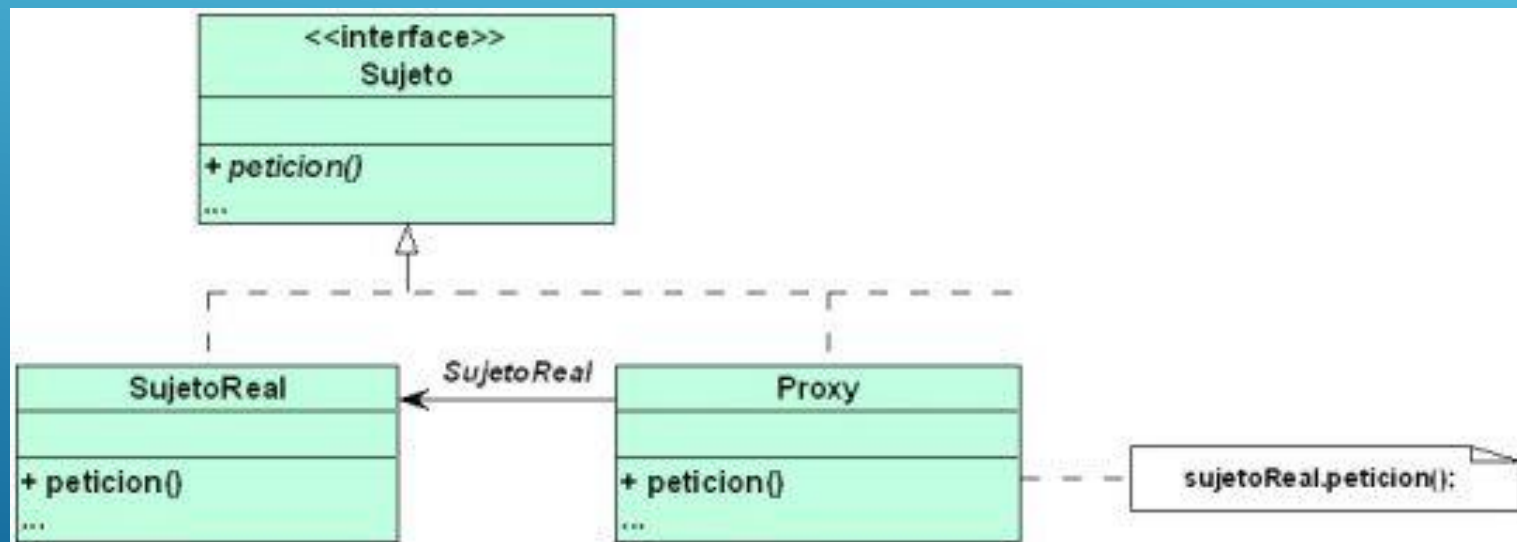
proxy remoto: representante local de un objeto remoto.

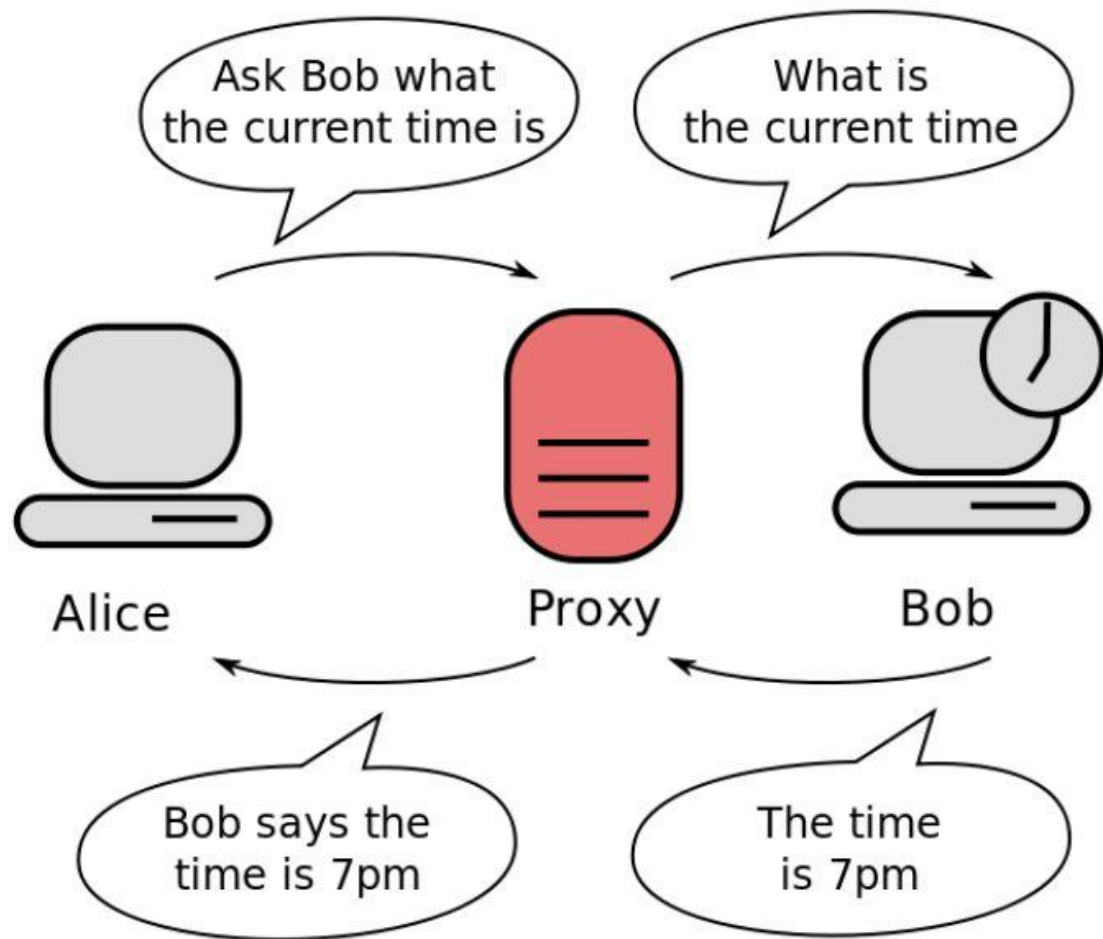
proxy virtual: crea objetos costosos bajo demanda

proxy de protección: controla el acceso al objeto original

proxy de referencia inteligente: sustituto de un puntero que lleva a cabo operaciones adicionales cuando se accede a un objeto

ESTRUCTURA



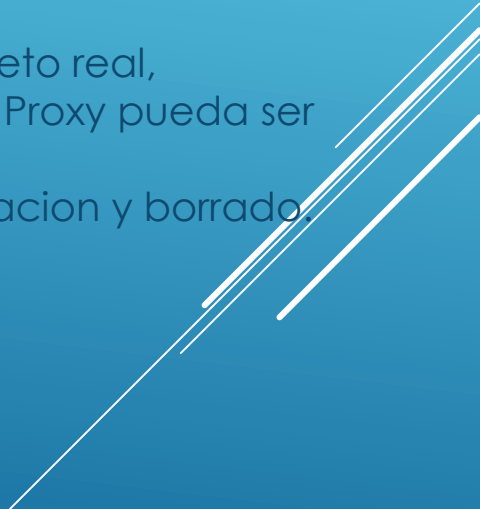


PARTICIPANTES

Subject: Define la interfaz común para el RealSubject y el proxy, de modo que pueda usarse un Proxy en cualquier sitio en el que se espere un RealSubject.


RealSubject: Define el objeto real representado.

Proxy: Mantiene una referencia que permite al Proxy acceder al objeto real, Proporciona una interfaz idéntica a la del sujeto, de manera que un Proxy pueda ser sustituido por el sujeto real.
Controla el acceso al sujeto real, y puede ser responsable de su creación y borrado.



COLABORACIONES

Proxy envía peticiones a
RelaSubject cuando es
apropiado, dependiendo del
tipo de proxy.


Several white lines of varying lengths and slopes are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.

CONSECUENCIAS

- ↓ El patrón de diseño proxy introduce cierto rango de desvío sobre el acceso de un objeto, los usos de este desvío dependen del tipo de proxy.
- ↓
 - Un proxy remoto oculta el hecho que objetos residen en diferentes espacios de direcciones.
- ↓
 - Un proxy virtual tales como crear o copiar un objeto bajo demanda.
- ↓
 - Un proxy para protección o las referencias inteligentes permiten realizar tareas de control sobre los objetos accedidos.

IMPLEMENTACIÓN

Este patrón es aconsejable en los siguientes supuestos:

- ↴ Retrasar una invocación a una petición con alto coste computacional hasta que sea necesario (*lazy loading*).
 - ↴ Simplificar la interacción con elementos remotos: si el objeto es local, las invocaciones serán las habituales. Si el objeto es remoto, el proxy implementará los mecanismos de comunicación haciéndolos transparentes al cliente.
- 
- A series of four parallel white diagonal lines of varying lengths, starting from the right edge and extending towards the bottom left, serving as a decorative element.

EJEMPLO DE CÓDIGO

```
"""
```

Proporcionar un sustituto o marcador de posición para otro objeto para controlar el acceso a este o agregar otras responsabilidades.

```
"""
```

```
import abc
```

```
class Subject(metaclass=abc.ABCMeta):
```

```
"""
```

Define la interfaz común para RealSubject y proxy para que Proxy se pueda usar en cualquier lugar donde se espera un RealSubject.

```
"""
```

```
@abc.abstractmethod
```

```
def request(self):
```

```
pass
```

```
class Proxy(Subject):
```

```
"""
```

Mantiene una referencia que le permite al Proxy acceder al RealSubject, proporciona una interfaz idéntica a la original.

```
"""
```

```
def __init__(self, real_subject):  
    self._real_subject = real_subject
```

```
def request(self):
```

```
# ...
```

```
self._real_subject.request()
```

```
# ...
```

```
class RealSubject(Subject):
```

```
"""
```

Define el objeto real que representa el Proxy.

```
"""
```

```
def request(self):
```

```
pass
```

```
def main():
```

```
    real_subject = RealSubject()
```

```
    proxy = Proxy(real_subject)
```

```
    proxy.request()
```

```
if __name__ == "__main__":  
    main()
```

USOS CONOCIDOS

A continuación se presentan algunos de los ejemplos más comunes en los que se utiliza el patrón *proxy* :

- ↓ Añadir acceso de seguridad a un objeto existente.
- ↓ Proporcionando interfaz de recursos remotos como el servicio web o recursos REST.
- ↓ Coordinación de las operaciones costosas en recursos remotos pidiendo los recursos a distancia para iniciar la operación tan pronto como sea posible antes de acceder a los recursos.
- ↓ Agregar una operación segura para los subprocesos a una clase existente sin cambiar el código de la clase existente.

PATRONES RELACIONADOS

- ↴ El patrón Adaptador proporciona una interfaz diferente al objeto que adapta, mientras que el *proxy* tiene la misma interfaz, pero ambos redirigen la petición del cliente al verdadero sujeto que la ejecuta con la posibilidad de incorporar lógica adicional : comprobación de acceso, creación del sujeto real...
- ↴ El *Proxy* se puede diseñar de manera similar al patrón decorador, pero el propósito es diferente: el decorador añade responsabilidades a un objeto, el *proxy* sólo controla su acceso. Así, si el *proxy* no tiene una fuerte dependencia con el sujeto real y no tiene que instanciarlo, puede adoptar el mismo diseño que el decorador, y ser un *proxy* de cualquier sujeto.