# Proxy Pattern

## Introducción y nombre

Proxy, Estructural. Controla el acceso a un recurso.

## Intención

El patrón Proxy se utiliza como intermediario para acceder a un objeto, permitiendo controlar el acceso a él.

El patrón obliga que las llamadas a un objeto ocurran indirectamente a través de un objeto proxy, que actúa como un sustituto del objeto original, delegando luego las llamadas a los métodos de los objetos respectivos.

## También conocido como

Surrogate, Virtual Proxy.

## Motivación

Necesitamos crear objetos que consumen muchos recursos, pero no queremos instanciarlos a no ser que el cliente lo solicite o se cumplan otras condiciones determinadas.

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

## Solución

Este patrón se debe utilizar cuando:

Se necesite retrasar el coste de crear e inicializar un objeto hasta que es realmente necesario.

Se necesita una referencia a un objeto más flexible o sofisticada que un puntero.

Algunas situaciones comunes de aplicación son:

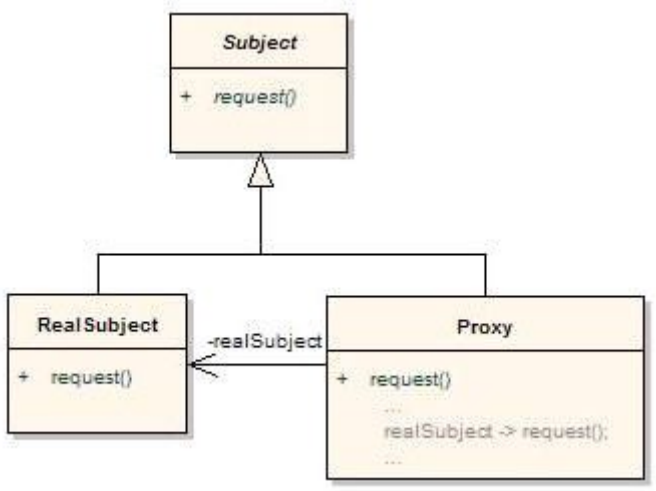
Proxy remoto: representa un objeto en otro espacio de direcciones. Esto quiere decir que el proxy será utilizado de manera tal que la conexión con el objeto remoto se realice de forma controlada sin saturar el servidor.

Proxy virtual: crea objetos costosos por encargo. Cuando se utiliza un software no siempre se cargan todas las opciones por default. Muchas veces se habilitan ciertos módulos sólo cuando el usuario decide utilizarlos.

Proxy de protección: controla el acceso a un objeto. Controla derechos de acceso diferentes.

Referencia inteligente: sustituto de un puntero que lleva a cabo operaciones adicionales cuando se accede a un objeto (ej. contar el número de referencias, cargar un objeto persistente en memoria, bloquear el objeto para impedir acceso concurrente, ...).

## Diagrama UML




# Participantes

Subject: interfaz o clase abstracta que proporciona un acceso común al objeto real y su representante (proxy).

Proxy: mantiene una referencia al objeto real. Controla la creación y acceso a las operaciones del objeto real.

RealSubject: define el objeto real representado por el Proxy.

# Colaboraciones

Cliente: solicita el servicio a través del Proxy y es éste quién se comunica con el RealSubject.

# Consecuencias

Introduce un nivel de dirección con diferentes usos:

Un proxy remoto puede ocultar el hecho de que un objeto reside en otro espacio de direcciones.

Un proxy virtual puede realizar optimizaciones, como la creación de objetos bajo demanda.

Los proxies de protección y las referencias inteligentes permiten realizar tareas de mantenimiento adicionales al acceder a un objeto.

Copiar un objeto grande puede ser costoso. Si la copia no se modifica, no es necesario incurrir en dicho gasto.

# Implementación

Tenemos un objeto padre Subject del que heredan: RealSubject y Proxy, todos ellos tienen un método request(). El cliente llamaría al método request () de Subject, el cual pasaría la petición a Proxy, que a su vez instanciaría RealSubject y llama a su request().

Esto nos permite controlar las peticiones a RealSubject mediante el Proxy, por ejemplo instanciando RealSubject cuando sea necesario y eliminándolo cuando deje de serlo.

# Código de muestra

Vamos a realizar un ejemplo de un proxy remoto: la finalidad es que nuestra aplicación guarde datos en un servidor remoto, pero vamos a impedir se la aplicación se conecte todo el tiempo, sino que aproveche a guardar todo cuando se encuentre conectada. Caso contrario guardará en el disco duro local la información hasta que sea el momento adecuado.

[code]

```
// Esta clase deberia ser un singleton
public class ConnectionManager {

    private static boolean hayConexion;

    public ConnectionManager() {
        hayConexion = false;
    }

    public static void conectate() {
        // Se abre la conexion
        hayConexion = true;
    }

    public static void desconectate() {
        // Se cierra la conexion
        hayConexion = false;
    }

    public static boolean hayConexion() {
        return hayConexion;
    }
}

public interface IGuardar {
    public void save(List datosAGuardar);
}

public class GuardarDiscoDuro implements IGuardar {
    @Override
    public void save(List datosAGuardar) {
        System.out.println("Guardando datos en el HD...");
    }
}

public class ObjetoRemoto implements IGuardar {
    @Override
    public void save(List datosAGuardar) {
        System.out.println("Guardando datos en el objeto remoto...");
    }
}

public class GuardarDatos implements IGuardar {
    @Override
    public void save(List datosAGuardar) {
        if (ConnectionManager.hayConexion()) {
            new ObjetoRemoto().save(datosAGuardar);
        } else {
            new GuardarDiscoDuro().save(datosAGuardar);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        List<String> datos = new ArrayList<String>();
        datos.add("Datos a guardar!!");

        IGuardar g = new GuardarDatos();
        g.save(datos);

        ConnectionManager.conectate();
        g.save(datos);
    }
}
[/code]
```

# Cuándo utilizarlo

El patrón Proxy es muy versátil. Puede ser utilizado en infinitas ocasiones y se le puede otorgar varios usos. Tiene una gran ventaja y es que no obliga al desarrollador a crear demasiada estructura para realizar este patrón, sino que es una forma estándar de acceder a una clase que potencialmente puede ser conflictiva. Por otro lado, no ayuda al desarrollador a crear un algoritmo, sino que el desarrollador tiene que hacer toda la lógica.

Por estas razones, es un patrón donde no siempre se puede saber a priori cuando utilizarlo. Sin embargo, un Proxy es un concepto utilizado fuera del ámbito de los patrones de diseño: un servidor proxy es un equipo intermediario situado entre el sistema del usuario e Internet. Puede utilizarse para registrar el uso de Internet y también para bloquear el acceso a una sede Web. El servidor de seguridad del servidor proxy bloquea algunas redes o páginas Web por diversas razones. En consecuencia, es posible que no pueda descargar el entorno de ejecución de Java (JRE) o ejecutar algunos applets de Java.

Es decir, los servidores proxy funcionan como filtros de contenidos. Y también mejoran el rendimiento: guardan en la memoria caché las páginas Web a las que acceden los sistemas de la red durante un cierto tiempo. Cuando un sistema solicita la misma página web, el servidor proxy utiliza la información guardada en la memoria caché en lugar de recuperarla del proveedor de contenidos. De esta forma, se accede con más rapidez.

Este mismo concepto se intenta llevarlo a cabo a nivel código con el patrón Proxy. Cuando un objeto debe ser controlado de alguna manera, ya sea por simple control de acceso o por estar en un sitio remoto o por ser muy pesado y se quiera limitar su uso, es ideal utilizar este patrón.

# Patrones relacionados

Facade: en ciertos casos puede resultar muy similar al facade, pero con mayor inteligencia.

Decorator: ciertas implementaciones se asemejan al decorador.

Singleton: la mayoría de las clases que hacen de proxy son Singletons.