Análisis y diseño de Sistemas

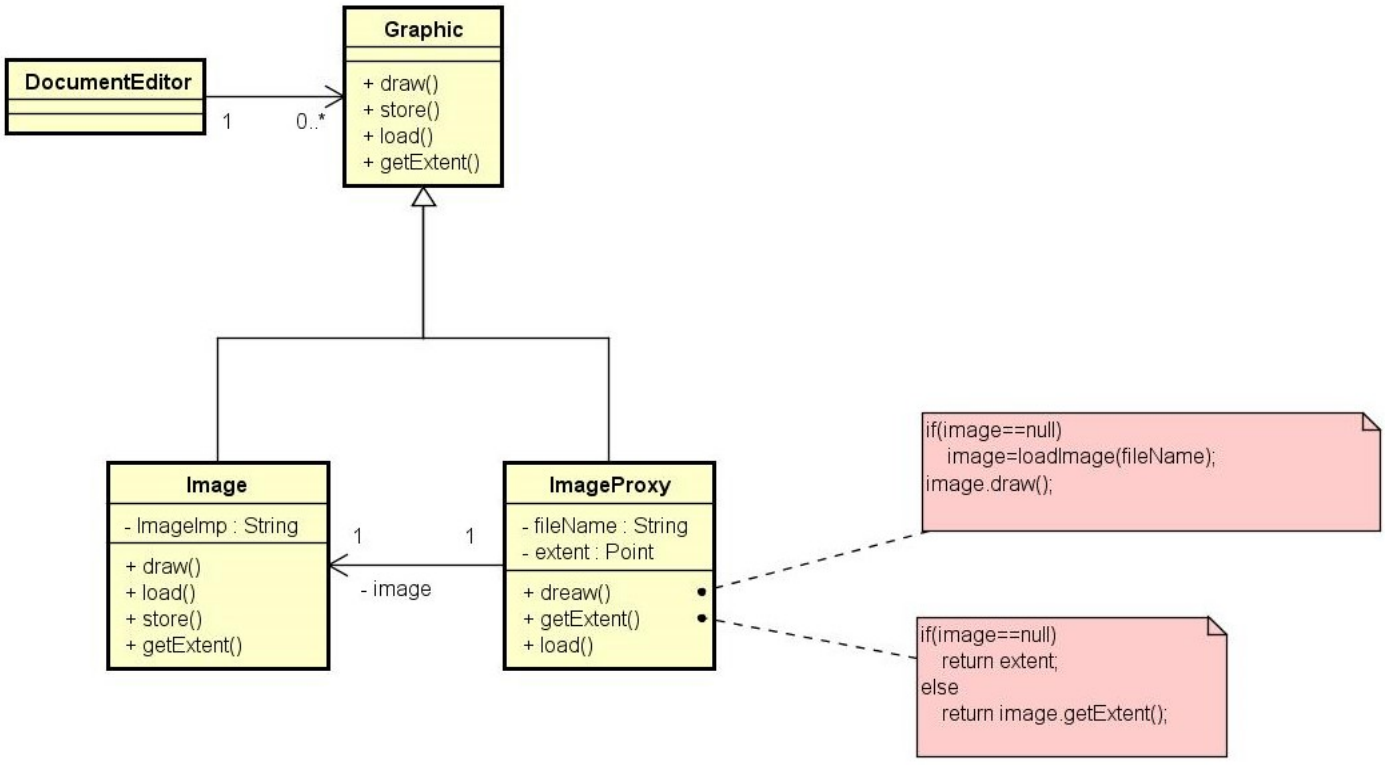
Patrones de diseño
Proxy

Matías Brizzio
Isaías Battaglia

Propósito y Motivación.

- **Propósito:** Proporcionar un representante de un objeto para controlar el acceso al mismo.
- **Motivación:** Retrasar la creación e inicialización de un objeto hasta que realmente sea necesario utilizarlo.

Motivación


Graphic
+ draw()
+ store()
+ load()
+ getExtent()
DocumentEditor
1
0..*
Image
- ImageImp : String
+ draw()
+ load()
+ store()
+ getExtent()
1
1
- image
ImageProxy
- fileName : String
- extent : Point
+ dreaw()
+ getExtent()
+ load()
if(image==null)
image=loadImage(fileName);
image.draw();
if(image==null)
return extent;
else
return image.getExtent();

Código

```java
public interface Image
  {public void displayImage();}
```

```java
public class RealImage implements Image {
  private String filename;

  public RealImage(String filename)
  {this.filename = filename; System.out.println("Cargando    "+filename);}

  public void displayImage() { System.out.println("Mostrando "+filename); }
}
```

```java
public class ProxyImage implements Image {
  private String filename;
  private RealImage image;

  public ProxyImage(String filename)
    {this.filename = filename;}

  public void displayImage() {
    if (image == null)
      image = new RealImage(filename); // carga imagen solo en demanda
    image.displayImage();
  }
}
```

```java
public class ProxyPattern{
  public static void main(String[] args) {
    Image image = new ProxyImage("prueba_10mb.jpg");
    Image image2 = new ProxyImage("prueba2_10mb.jpg");
    image.displayImage();
    image2.displayImage();
    System.out.println("");
    image.displayImage();
    image2.displayImage();
  }
}
```

Solución

- Estas restricciones sugieren que cada objeto se cree a petición, lo que en este caso tendrá lugar cuando la imagen se hace visible.

  La solución es utilizar otro objeto, un *proxy* que actúe como un sustituto de la imagen real.

Estructura

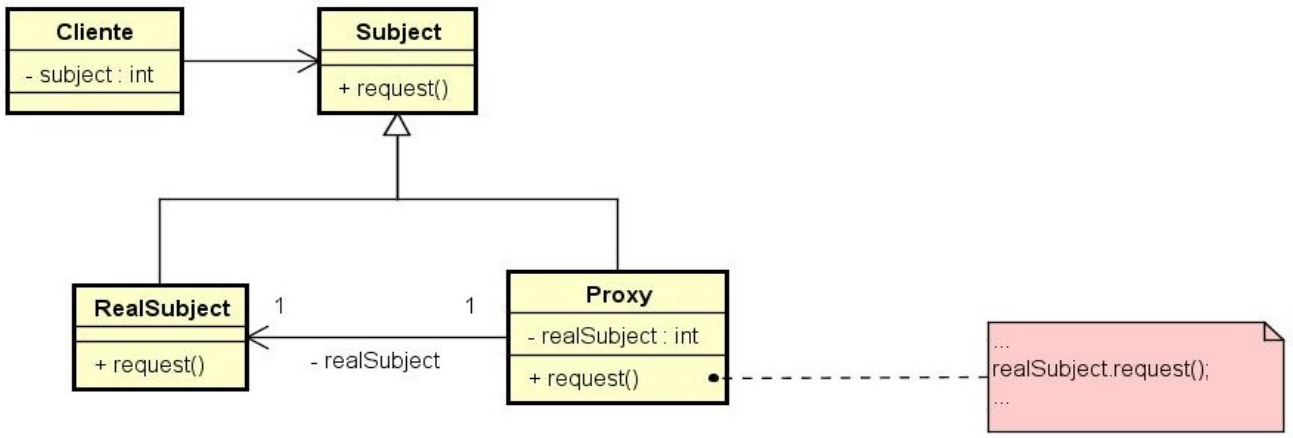
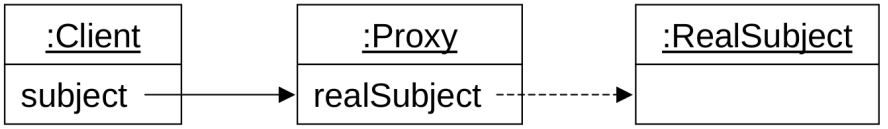


Diagrama de objetos



Aplicabilidad

El patrón Proxy se usa cuando 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.
- Proxy Virtual: Crea objetos costosos por demanda.
- Proxy Protección: Controla el acceso a un objeto y es útil cuando un objeto tiene diferentes derechos de acceso.

Participantes

Los participantes en el patron de diseño Proxy son:

- **Proxy**
  - Mantiene una referencia al objeto real.
  - Provee una interfaz identica al objeto.
  - Controla el acceso al objeto real, y puede ser el encargado de crearlo y borrarlo.
  - *Otras responsabilidades que dependen de la clase proxy con la que se trabaje son:*
    - ***Proxies remotos:*** Codifican las peticiones y sus argumentos para enviarlas al objeto real.
    - ***Proxies virtuales:*** Almacena informacion adicional del objeto real para posponer el acceso a este.
    - ***Proxies de proteccion:*** Chequean que el invocador tenga permisos para realizar la peticion.
- **Subject (sujeto):** Define una interfaz comun al objeto real y al proxy.
- **Real Subject (sujeto real):** Define el objeto real, el cual es representado por el proxy.

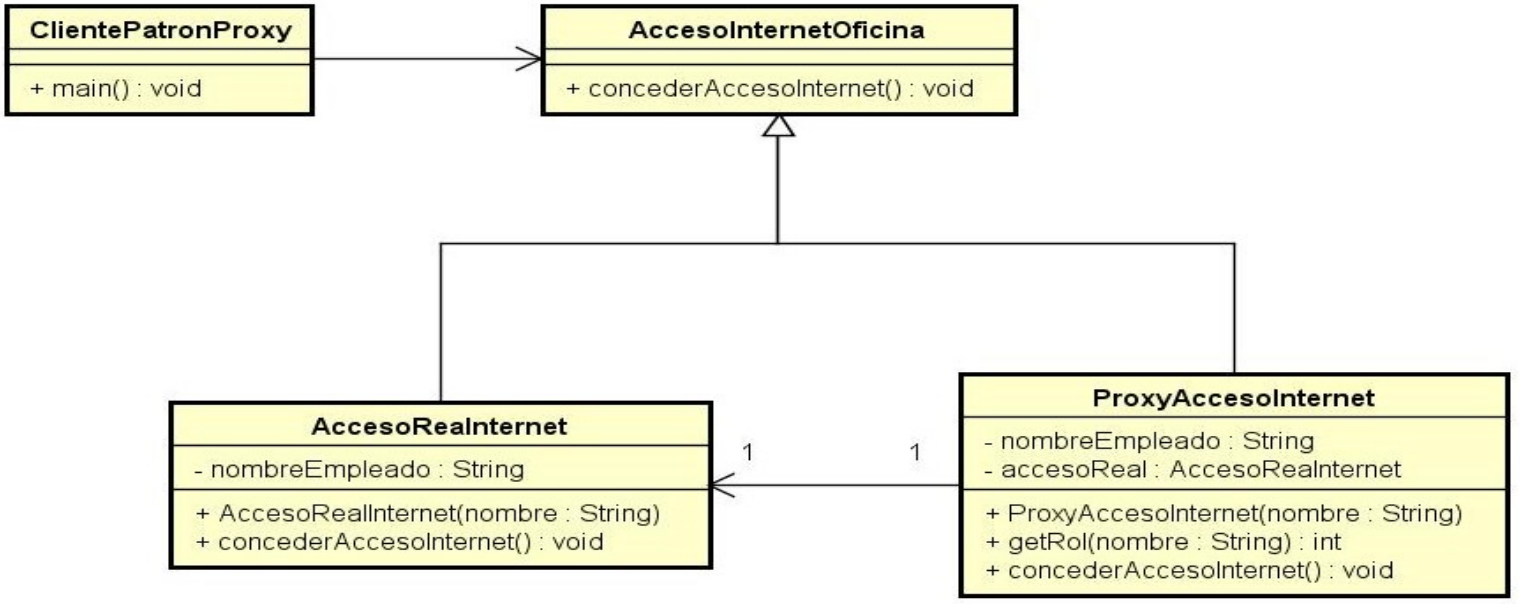
Consecuencias

- ***Ventajas***
- Introduce un nivel de indirección cuando se accede al objeto, este nivel posee diferentes usos dependiendo de la clase de proxy:
  - ***Proxy Remoto***: Puede ocultar el hecho de que un objeto reside en diferentes espacios de direcciones.
  - ***Proxy virtual:*** Puede realizar optimizaciones, como la creación de objetos bajo demanda
  - ***Proxies de protección y referencias inteligentes:*** Permiten realizar tareas de mantenimiento adicionales al acceder a un objeto
- Otra optimización que se puede realizar es la denominada, Optimización ***copy-on-write***, su idea es la siguiente
  - Esta relacionada con la creación bajo demanda.
  - Copiar un objeto grande puede ser costoso
  - Si la copia no se modifica, no es necesario incurrir en dicho gasto
  - El sujeto mantiene un número de referencias, y sólo cuando se realiza una operación que modifica el objeto, éste se copia
  - Es útil por tanto para retrasar la replicación de un objeto hasta que cambia,
- ***Desventajas:***
  - Complejas implementación

Ejemplo

- Un ejemplo adicional:
- **Proxy de protección:**
  - Un servidor proxy que proporciona una restricción en el acceso a Internet en una oficina. Sólo se permitirán los accesos a determinados sitios web y los contenidos a cierto tipo de empleados.

Diagrama UML Ejemplo



Código

```java
public interface AccesoInternetOficina
    {public void concederAccesoInternet();}
```

```java
public class AccesoRealInternet implements AccesoInternetOficina{
  private String nombreEmpleado;

  public AccesoRealInternet(String empName)
    {this.nombreEmpleado = empName;}

  @Override
  public void concederAccesoInternet()
    {System.out.println("Acceso a internet concedido para: "+ nombreEmpleado);}
}
```

```java
public class ProxyAccesoInternet implements AccesoInternetOficina{

  private String nombreEmpleado;
  private AccesoRealInternet  accesoReal;

  public ProxyAccesoInternet(String nombreEmpleado)
    {this.nombreEmpleado = nombreEmpleado;}

  @Override
  public void concederAccesoInternet(){
    if (getRol(nombreEmpleado)>4){
      accesoReal = new AccesoRealInternet(nombreEmpleado);
      accesoReal.concederAccesoInternet();
    }
    else
      System.out.println("Acceso a internet no concedido.");
  }

  public int getRol(String emplName){
  /*Este metodo podria chequear el rol del empleado en una base de datos
  y en base a ello conceder o no el permiso para acceder*/
    if (emplName.equals("Matias") || emplName.equals("Isaias")) return 5;
    return 2;
  }
}
```

```java
public class ClientePatronProxy{

  public static void main(String[] args){
    Empleado client = new Empleado("Matias");
    client.obtenerAccesoInternet();
  }

  private static class Empleado{
    private String nombre;
    public Empleado(String nombre)
      {this.nombre=nombre;}
    public void obtenerAccesoInternet(){
      AccesoInternetOficina acces = new ProxyAccesoInternet(nombre);
      acces.concederAccesoInternet();
    }
  }
}
```

Referencias

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software Addison-Wesley, 1995 1st Edition
- https://www.javatpoint.com/proxy-pattern
- https://en.wikipedia.org/wiki/Proxy_pattern