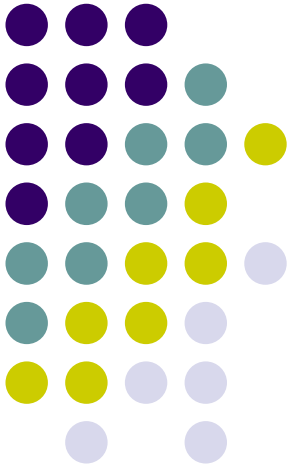
# Patrones de Diseño

## Patrón estructural *Proxy*



# *Proxy*

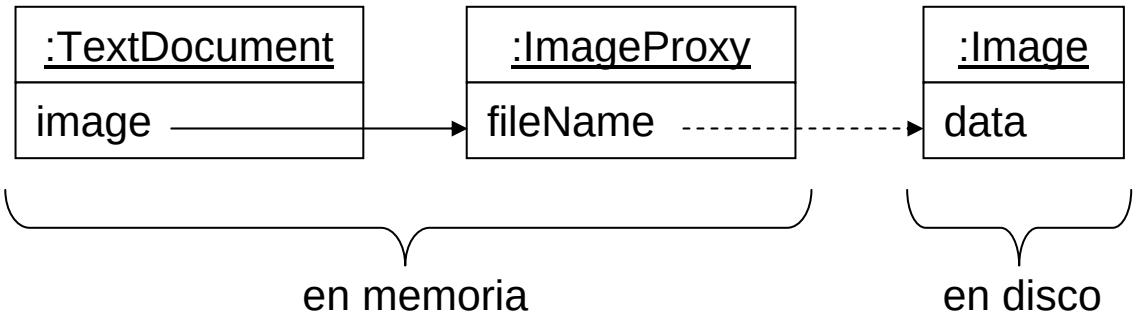
## Propósito


- Proporcionar un representante o sustituto de otro objeto para controlar el acceso a éste

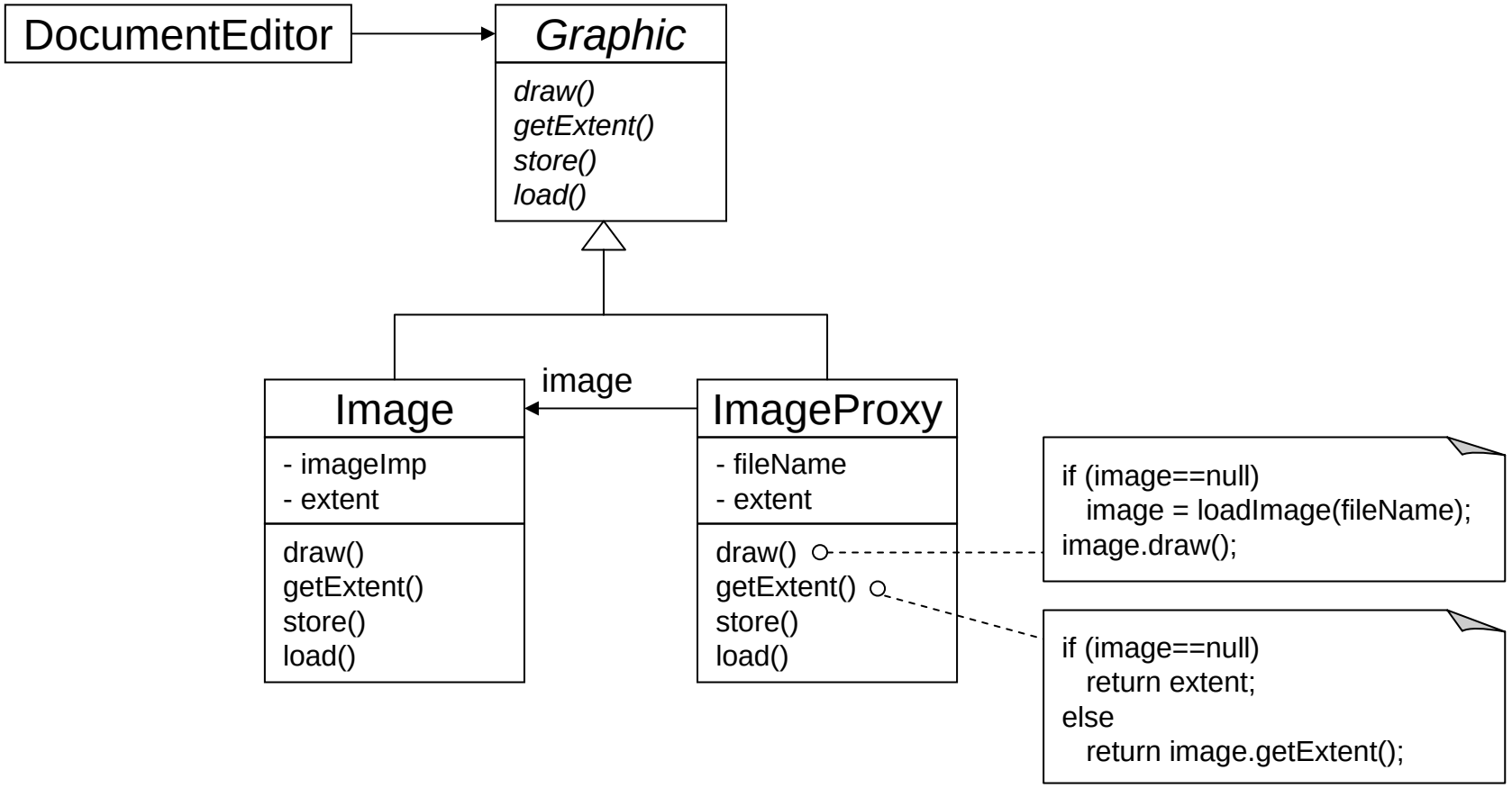
## Motivación

- 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

- Solución:

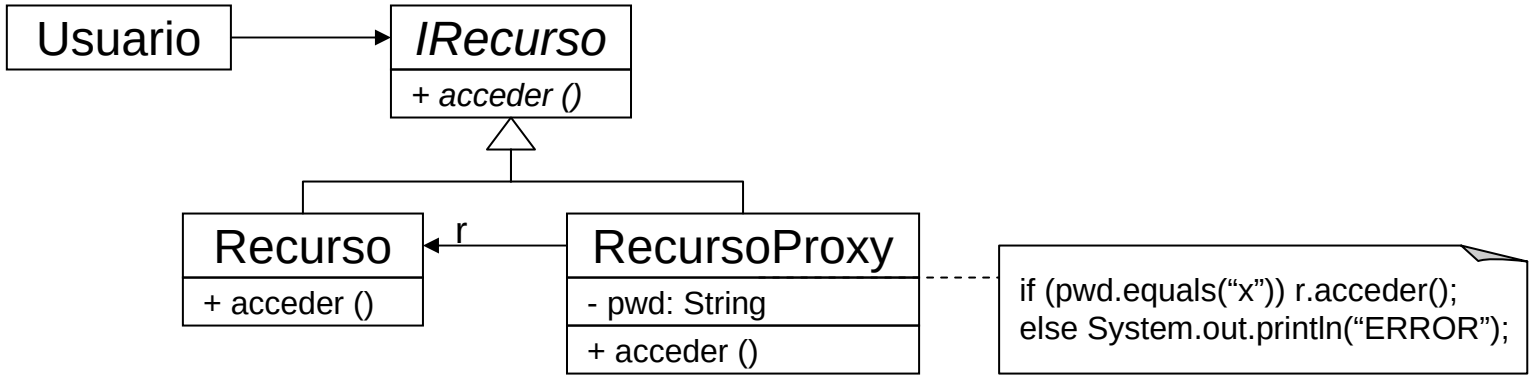


# *Proxy*
## Motivación



# *Proxy*
## Aplicabilidad


- El patrón *Proxy* se usa cuando se necesita una referencia a un objeto más flexible o sofisticada que un puntero. Por ejemplo:
  - **Proxy virtual**: crea objetos costosos por encargo (como la clase *ImageProxy* en el ejemplo de motivación)
  - **Proxy remoto**: representa un objeto en otro espacio de direcciones
  - **Referencia inteligente**: sustituto de un puntero que lleva a cabo operaciones adicionales cuando se accede a un objeto (ej. contar número de referencias, cargar un objeto persistente en memoria, bloquear el objeto para impedir acceso concurrente, ...)
  - **Proxy de protección**: controla el acceso a un objeto



# *Proxy*
## Estructura

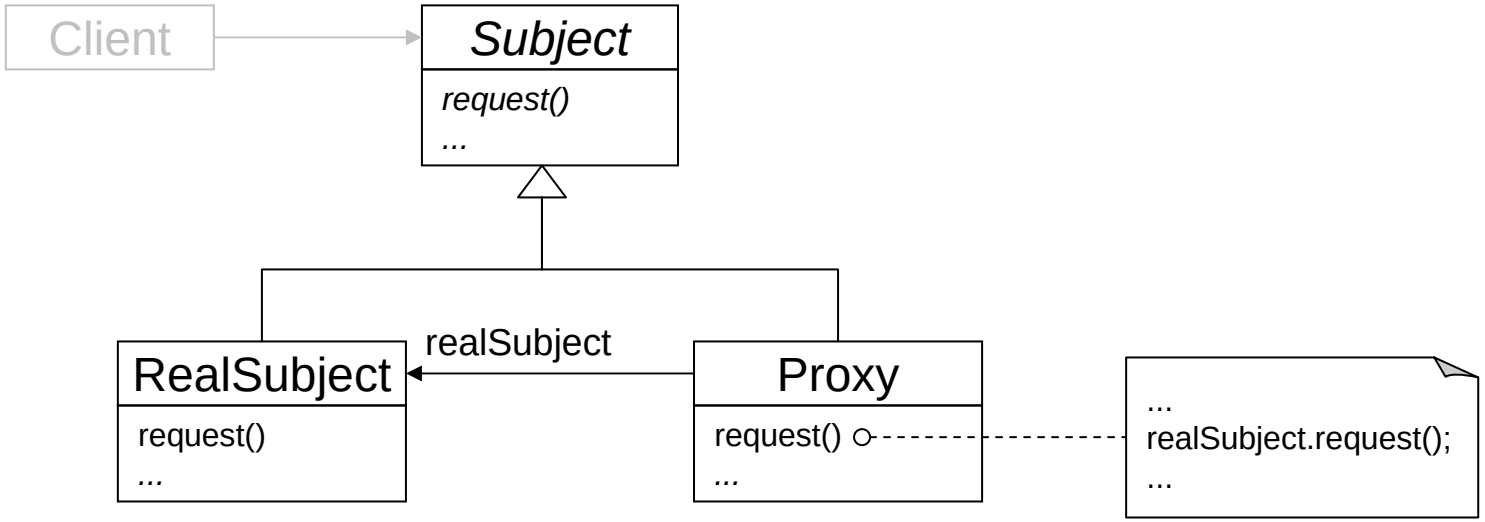
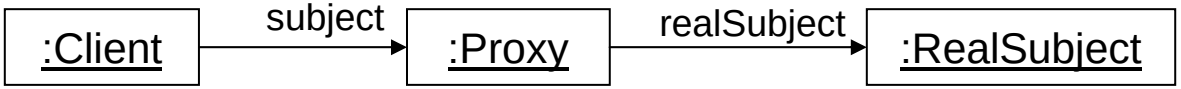



Diagrama de objetos



# *Proxy*
## Participantes


- **Proxy** (*ImageProxy*):
  - Mantiene una referencia al objeto real
  - Proporciona una interfaz idéntica a la del objeto real
  - Controla el acceso al objeto real, y puede ser responsable de crearlo y borrarlo
  - Otras responsabilidades dependen del tipo de proxy:
    - Proxies remotos: codifican las peticiones, y las envían al objeto real
    - Proxies virtuales: pueden guardar información del objeto real (caché)
    - Proxies de protección: comprueban que el cliente tiene los permisos necesarios para realizar la petición
- **Subject** (*Graphic*): define una interfaz común para el proxy y el objeto real, de tal modo que se puedan usar de manera indistinta
- **RealSubject** (*Image*): clase del objeto real que el proxy representa

# *Proxy*
## Consecuencias

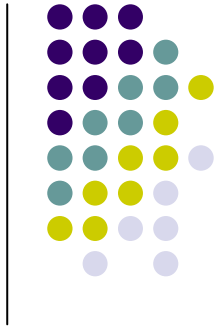

- Introduce un nivel de indirección con diferentes usos:
  - Un proxy remoto puede ocultar el hecho de que un objeto reside en otro espacio de direcciones
  - Un proxy virtual puede realizar optimizaciones, como la creación de objetos bajo demanda
  - Los proxies de protección y las referencias inteligentes permiten realizar tareas de mantenimiento adicionales al acceder a un objeto
- Optimización copy-on-write
  - 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

# *Proxy*
## Implementación (proxy virtual)


```
public abstract class Graphic {
   public void draw();
}


public class Image extends Graphic {
   public void draw() { ... }
}


public class ImageProxy extends Graphic {
   private Image  _image;
   private String _fileName;
   public ImageProxy (String fileName) {
      _fileName = fileName;
      _image    = null;
   }
   public Image loadImage() { ... }
   public void draw () {
      if (_image==null)
         _image = loadImage(_fileName);
      _image.draw();
   }
}
```

```
public class TextDocument {
   public void insert (Graphic g) { ... }
}


// código para insertar una ImageProxy
// en un documento
TextDocument td = new TextDocument();
Graphic g = new ImageProxy("imagen.gif");
td.insertar(g);
```

# Conclusiones



- Los patrones de diseño describen la solución a problemas que se repiten una y otra vez en nuestros sistemas, de forma que se puede usar esa solución siempre que haga falta
- Capturan el conocimiento que tienen los expertos a la hora de diseñar
- Ayudan a generar software “maleable” (software que soporta y facilita el cambio, la reutilización y la mejora)
- Son guías de diseño, no reglas rigurosas