

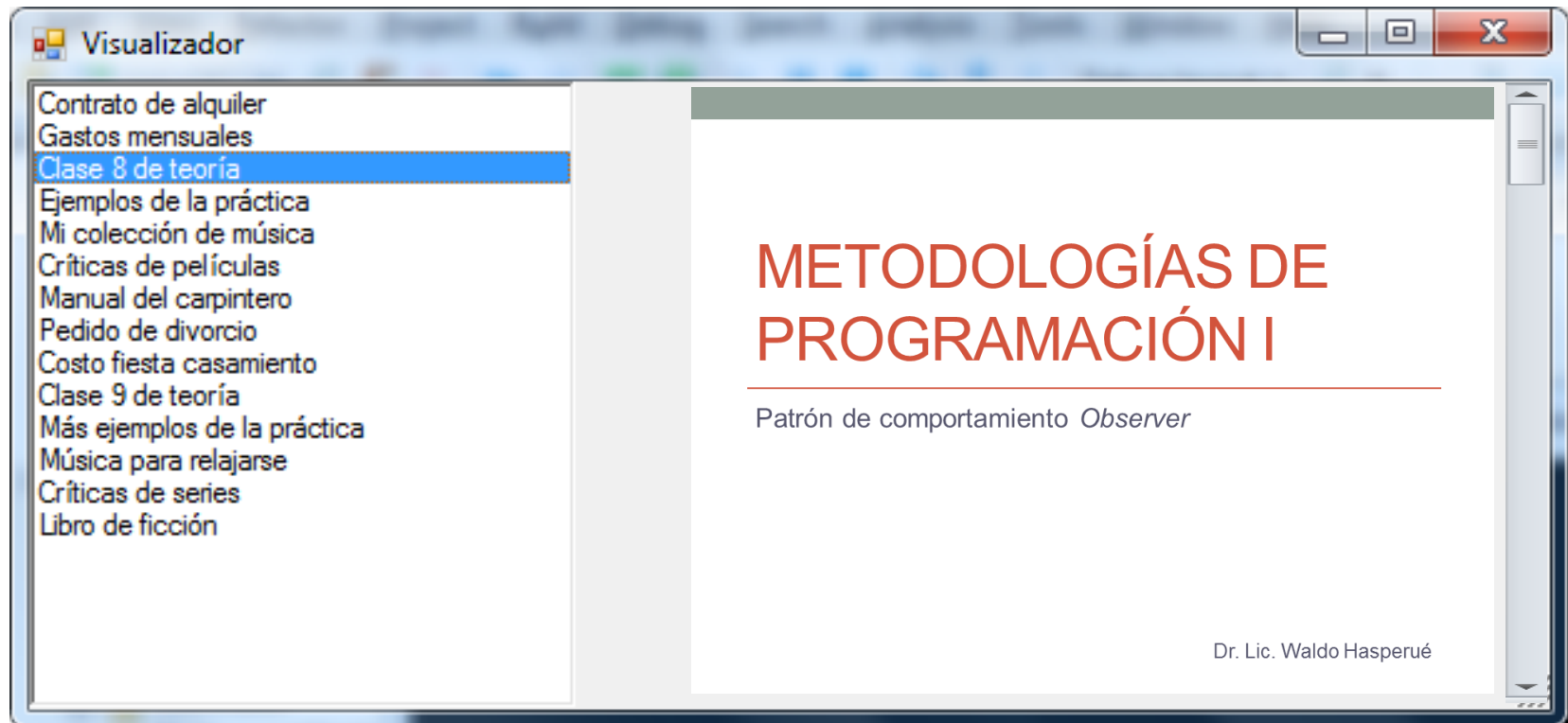
# METODOLOGÍAS DE PROGRAMACIÓN I

---

Patrón de comportamiento *Proxy*

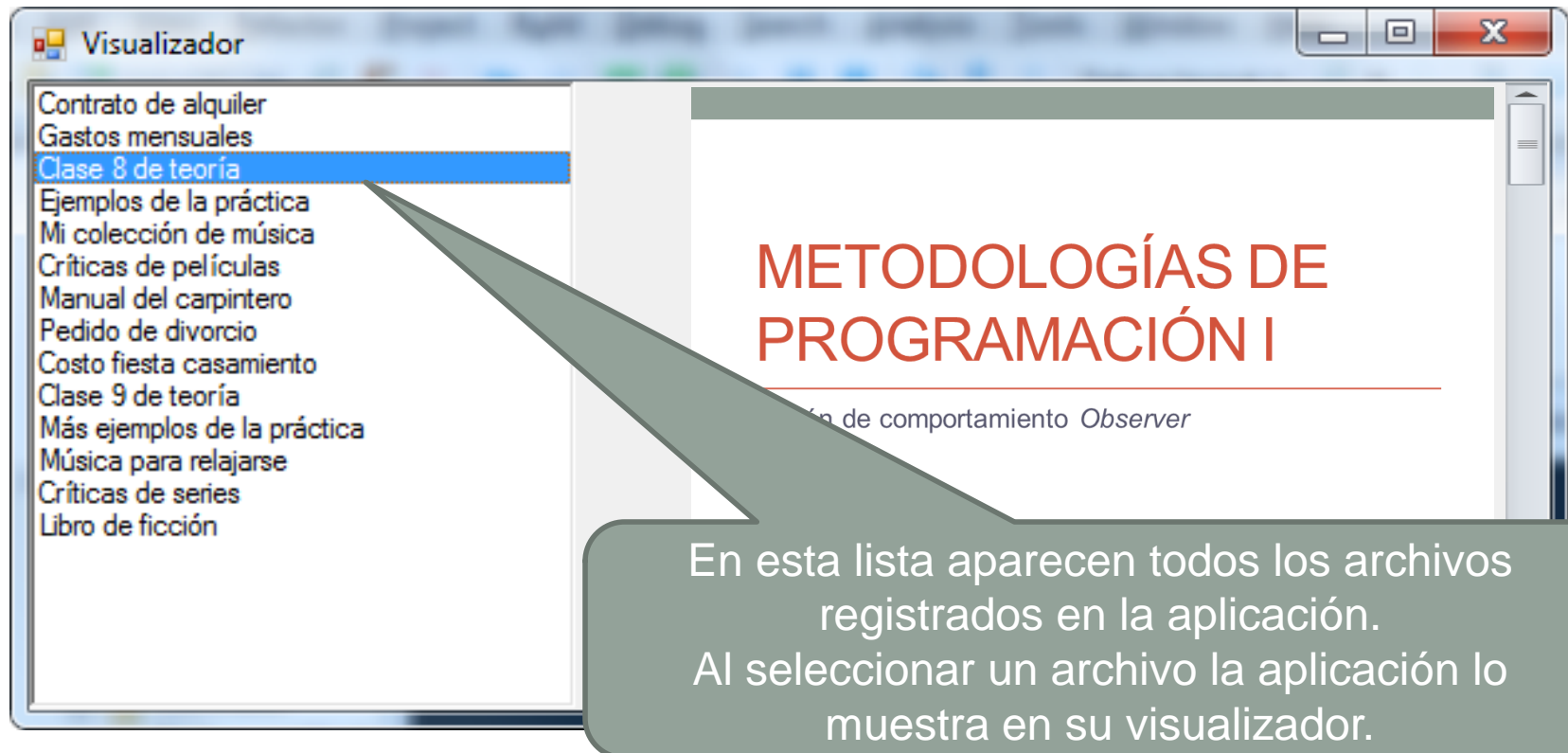
# Situación de ejemplo

- Se cuenta con una aplicación que es capaz de mostrar distintos tipos de documentos (documentos de texto, planillas de cálculo, presentación de diapositivas, etc.)



# Situación de ejemplo

- Se cuenta con una aplicación que es capaz de mostrar distintos tipos de documentos (documentos de texto, planillas de cálculo, presentación de diapositivas, etc.)



# Situación de ejemplo

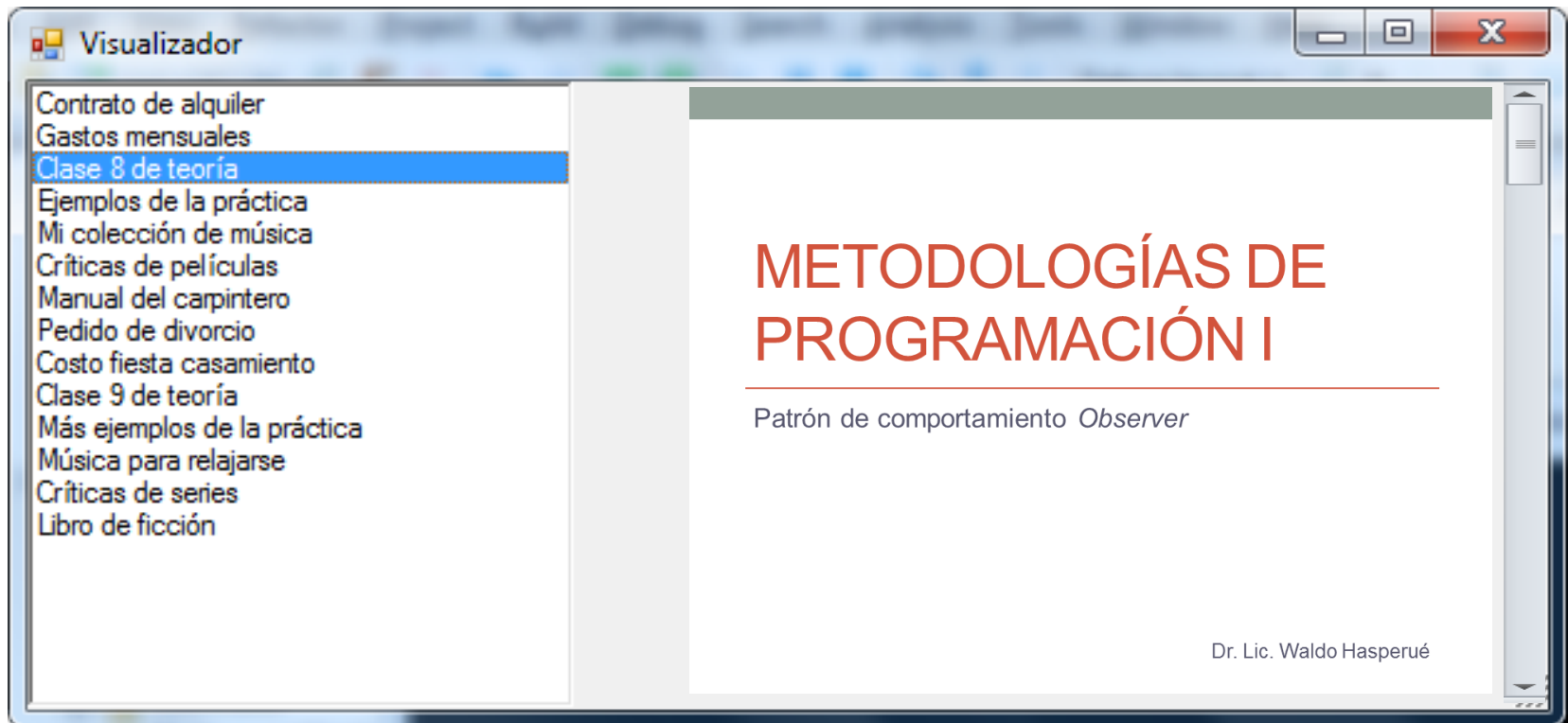
Esta aplicación cuenta con la siguiente interface:

```
interface IDocumento
    // Para mostrar el nombre del archivo en
    // la lista
    string getNombre()

    // Para mostrar la vista previa del
    // archivo seleccionado
    void mostrarVistaPrevia()
```

# Situación de ejemplo

Cuando la aplicación arranca, carga todos sus archivos registrados en la lista:



# Problema

- OK ¿Pero cuál es el problema?
- Al iniciar la aplicación crea todos los documentos de la lista:

```
void iniciarAplicacion (archivos)
    foreach a in archivos:
        lista.Add(fabrica.crearDocumento(a))
```

```
class FabricaDeDocumentos
    . . . new DocumentoDeTexto()
    . . . new PlanillaDeCalculo()
    . . . new PresentacionConDiapositivas()
    . . . new ImagenAdapter(new Imagen())
```

# Problema

- OK ¿Pero CUÁL es el problema?

```
class FabricaDeDocumentos
    . . . new DocumentoDeTexto()
    . . . new PlanillaDeCalculo()
    . . . new PresentacionConDiapositivas()
    . . . new ImagenAdapter(new Imagen())
```

- Al crear un documento se debe:
  - Leer todos los datos desde un archivo
  - Crear toda su estructura interna, la cual puede estar compuesta por muchos otros objetos
    - Documento de texto: objetos secciones, páginas, párrafos, configuraciones de página, figuras, tablas, índices, etc.
    - Planilla de cálculo: objetos hojas, celdas, fórmulas, tablas, gráficos, etc.
    - Presentación con diapositivas: objetos diapositivas, objetos de dibujos, cuadros de texto, animaciones, etc.

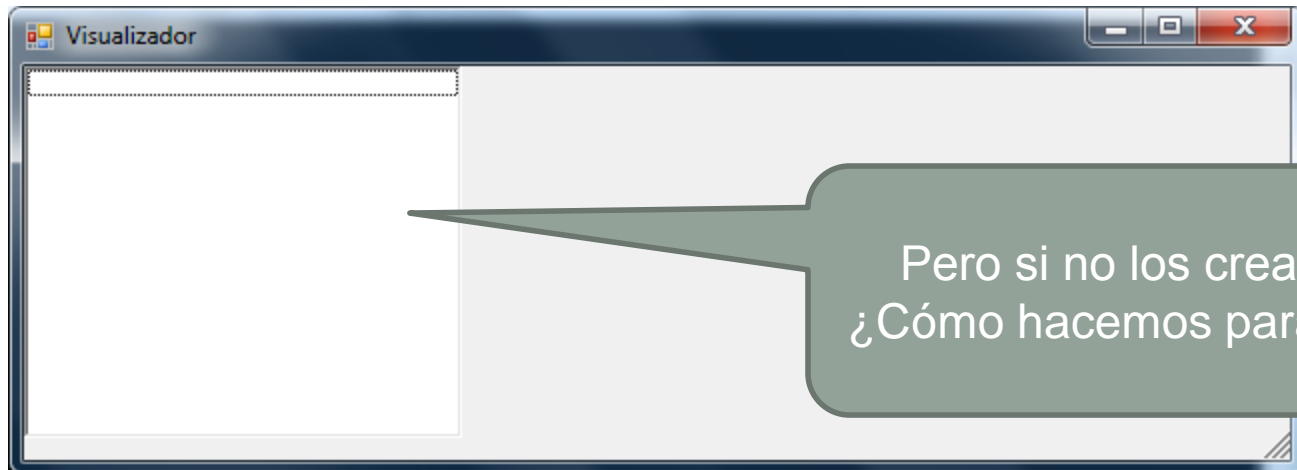
# Problema

- OK ¿Pero CUÁL es el problema?
- Para crear todos los documentos necesitamos:
  - Tiempo de carga (lectura desde archivo y creación del objeto)
  - Memoria para almacenar todos los documentos
  - Recursos del sistema operativo para leer y crear los documentos
- Ahora ¿Por qué perder tiempo, recursos y espacio en memoria para crear cientos o miles de documentos si el usuario de la aplicación solo quería consultar UNO solo?



# Problema

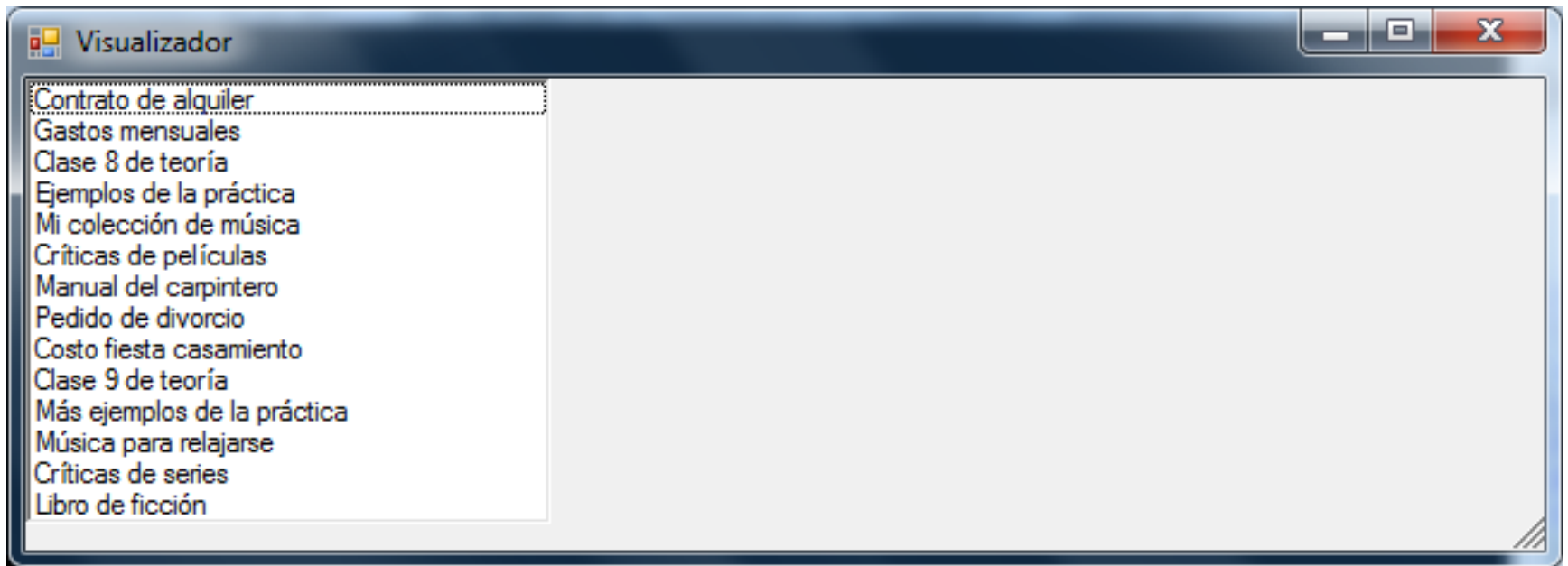
- OK ¿Pero CUÁL es el problema?
- Se desperdicia tiempo, recursos y espacio en memoria solo para (quizás) mostrar un único archivo.
- Entonces no creamos todos los objetos al inicio de la aplicación



# Problema

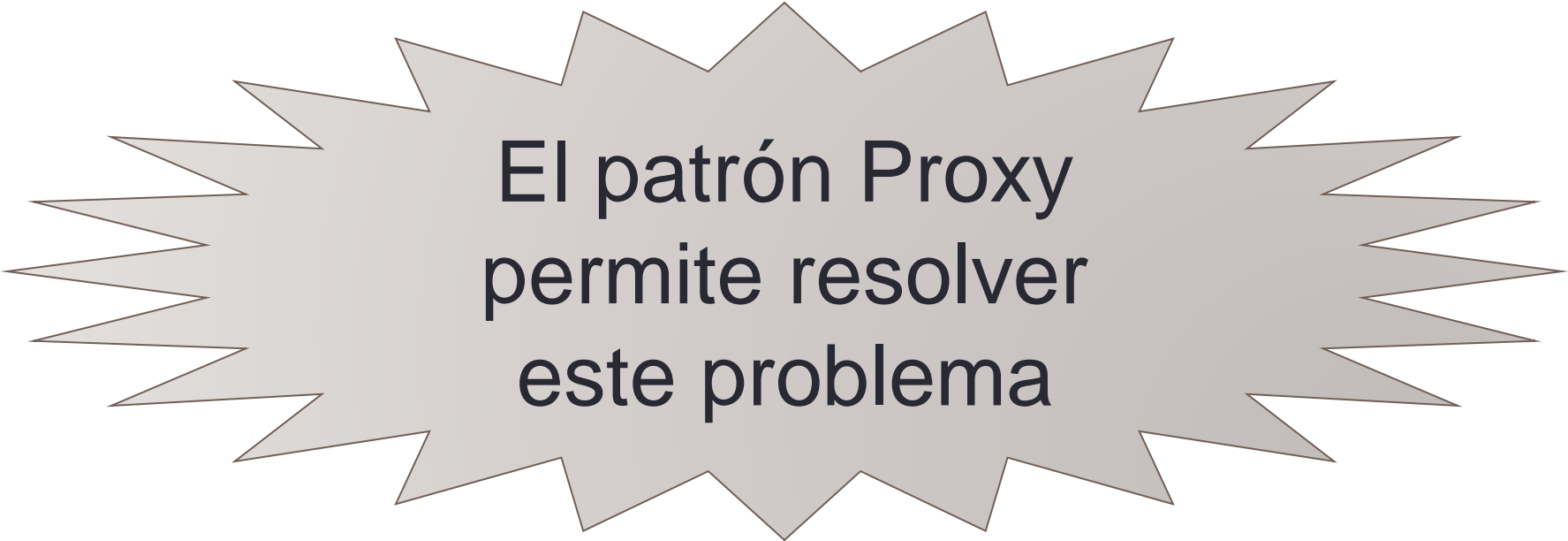
La pregunta definitiva sería:

¿cómo hacemos para crear objetos sin crearlos?



# Motivación

¿Es posible contar con un mecanismo que permita crear objetos con funcionalidades mínimas (mostrar su nombre en una lista) y solo cuando sea necesario agregarle su funcionalidad completa?



El patrón Proxy  
permite resolver  
este problema

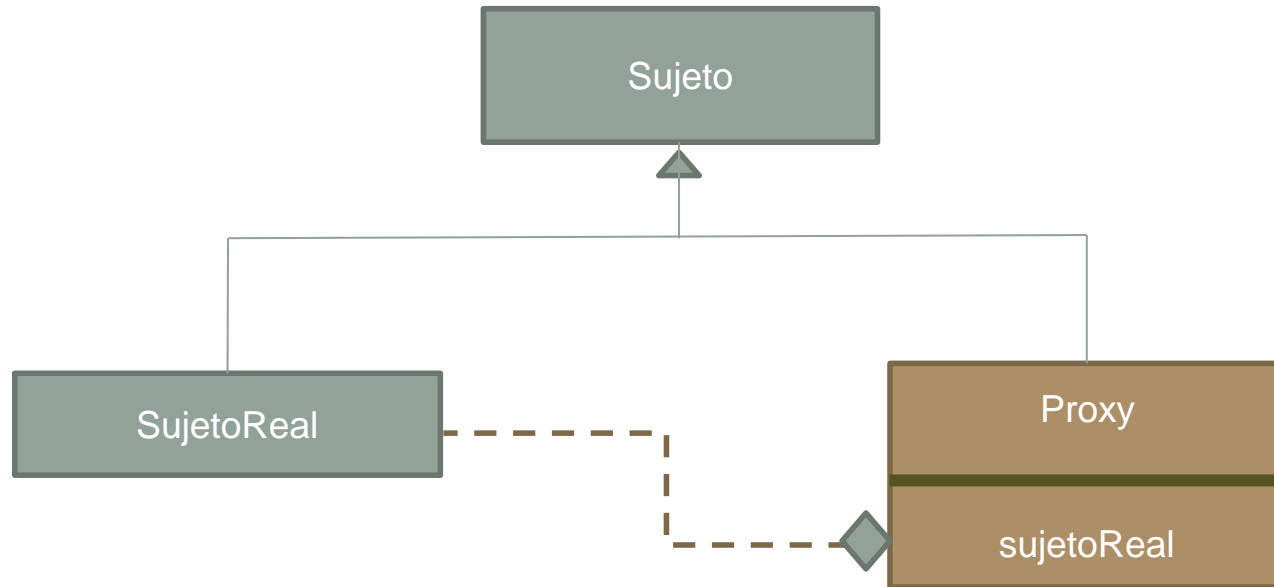
# Proxy

**Propósito:** Proporciona un representante o sustituto de otro objeto para controlar el acceso a este.

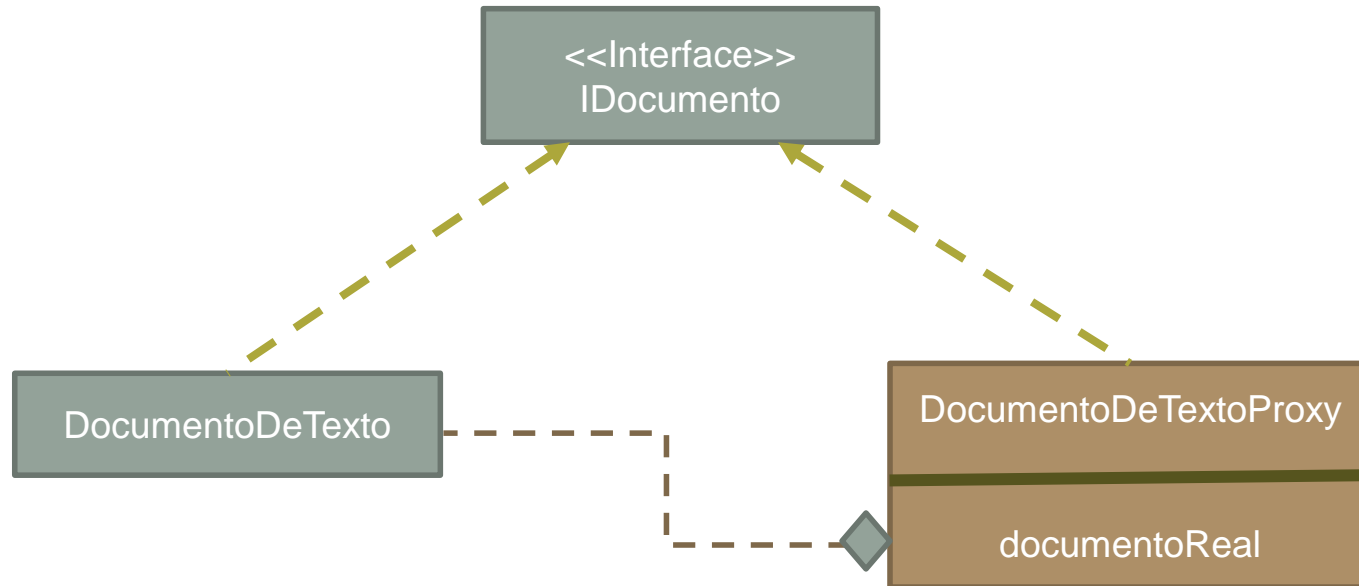
**Aplicabilidad:** usarlo cuando

- Necesite retrasar la carga de información desde bases de datos, archivos o recursos de red.
- Retrasar la construcción de grandes objetos costosos
- Necesite garantizar la protección de objetos
- Realizar operaciones adicionales con fines de optimización

# Proxy - Estructura



# Proxy - Estructura



# Proxy - Implementación

```
interface IDocumento
```

```
    string getNombre()
```

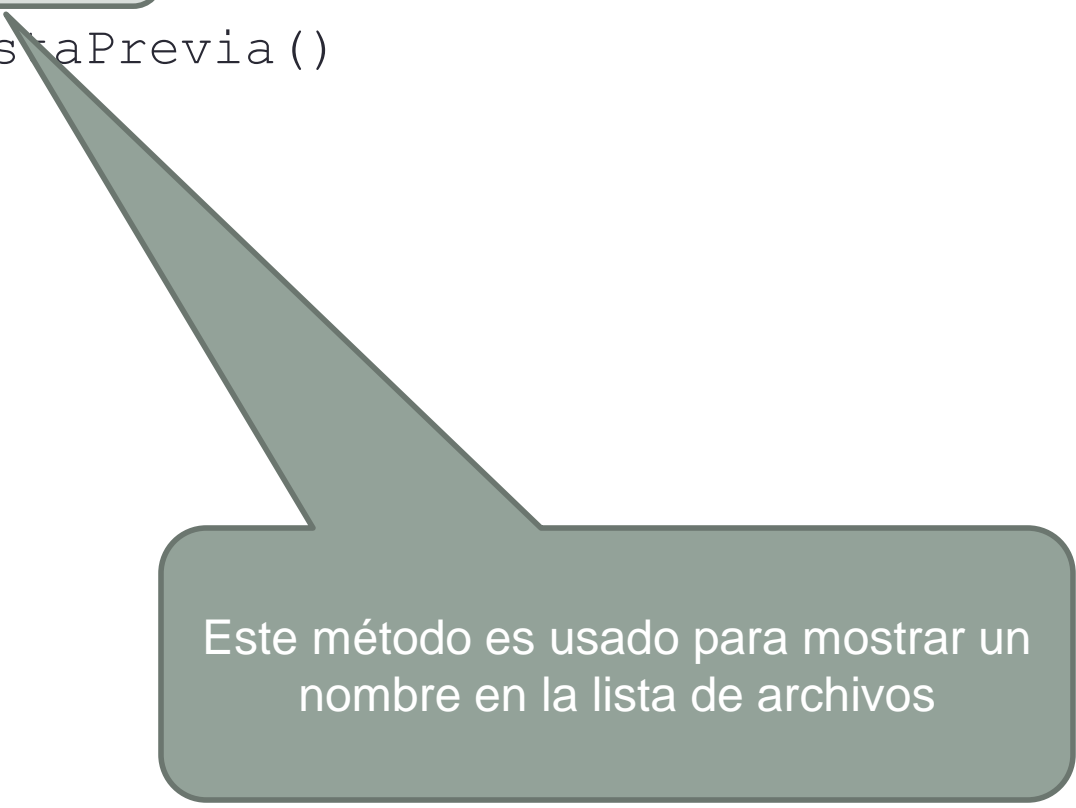
```
    void mostrarVistaPrevia()
```

# Proxy - Implementación

```
interface IDocumento
```

```
    string getNombre()
```

```
    void mostrarVistaPrevia()
```



Este método es usado para mostrar un nombre en la lista de archivos

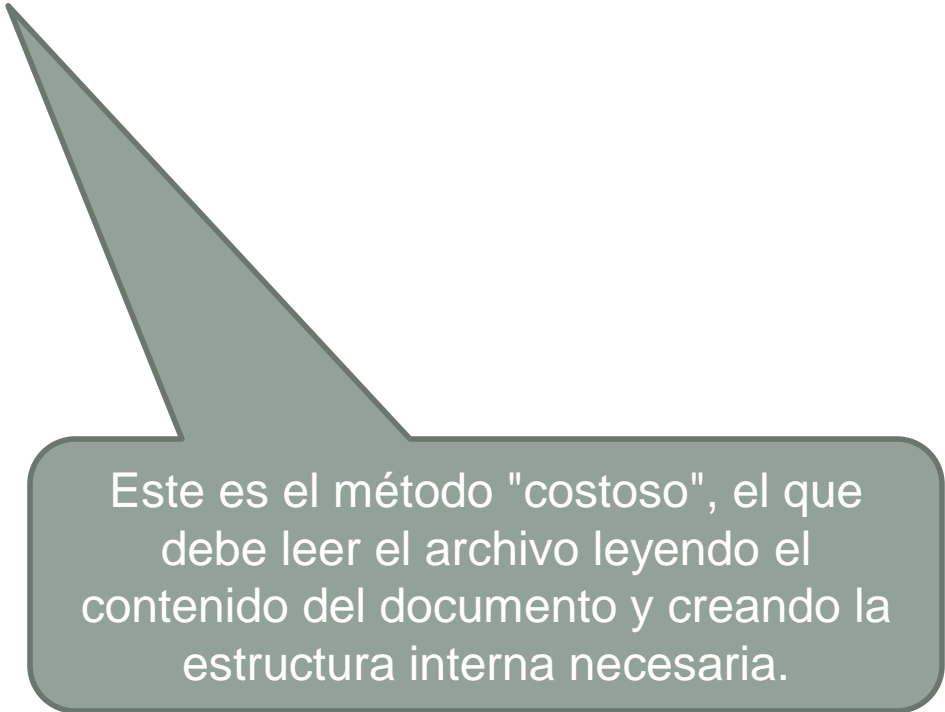


# Proxy - Implementación

```
interface IDocumento
```

```
    string getNombre()
```

```
    void mostrarVistaPrevia()
```



Este es el método "costoso", el que debe leer el archivo leyendo el contenido del documento y creando la estructura interna necesaria.

# Proxy - Implementación

```
class DocumentoDeTexto : IDocumento

    string nombre

    DocumentoTexto(string s)
        nombre = s
        self.abrirDocumento()

    string getNombre()
        return nombre

    void mostrarVistaPrevia()
        . . .

    void abrirDocumento()
        método "costoso"
```

Todos los documentos  
implementan la  
interface *IDocumento*

# Proxy - Implementación

```
class DocumentoDeTextoProxy : IDocumento

    DocumentoDeTexto documentoReal = null
    string nombre

    constructor(string n)
        nombre = n

    string getNombre()
        return nombre
```

Creamos el proxy  
para los documentos

# Proxy - Implementación

```
class DocumentoDeTextoProxy : IDocumento

    DocumentoDeTexto documentoReal = null
    string nombre

    constructor(string n)
        nombre = n

    string getNombre()
        return nombre

    void mostrarVistaPrevia()
        . . .
```

Un proxy es capaz de responder las peticiones "sencillas" o simples de resolver.

# Proxy - Implementación

```
class DocumentoDeTextoProxy : IDocumento
    . . .

    void mostrarVistaPrevia()
        if(documentoReal == null)
            documentoReal = new DocumentoDeTexto(nombre)

        documentoReal.mostrarVistaPrevia()
```

# Proxy - Implementación

```
class DocumentoDeTextoProxy : IDocumento
    . . .

void mostrarVistaPrevia()
    if(documentoReal == null)
        documentoReal = new DocumentoDeTexto(nombre)

    documentoReal.mostrarVistaPrevia()
```

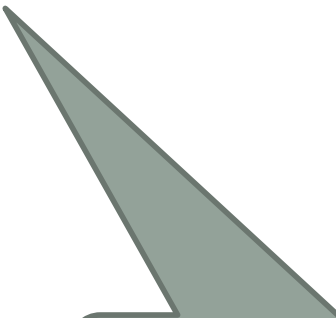
Cuando el proxy no puede sustituir al documento real se construye éste último, para que haga la tarea "costosa": leerse de un archivo y crear el objetos complejo

# Proxy - Implementación

```
class DocumentoDeTextoProxy : IDocumento
    . . .

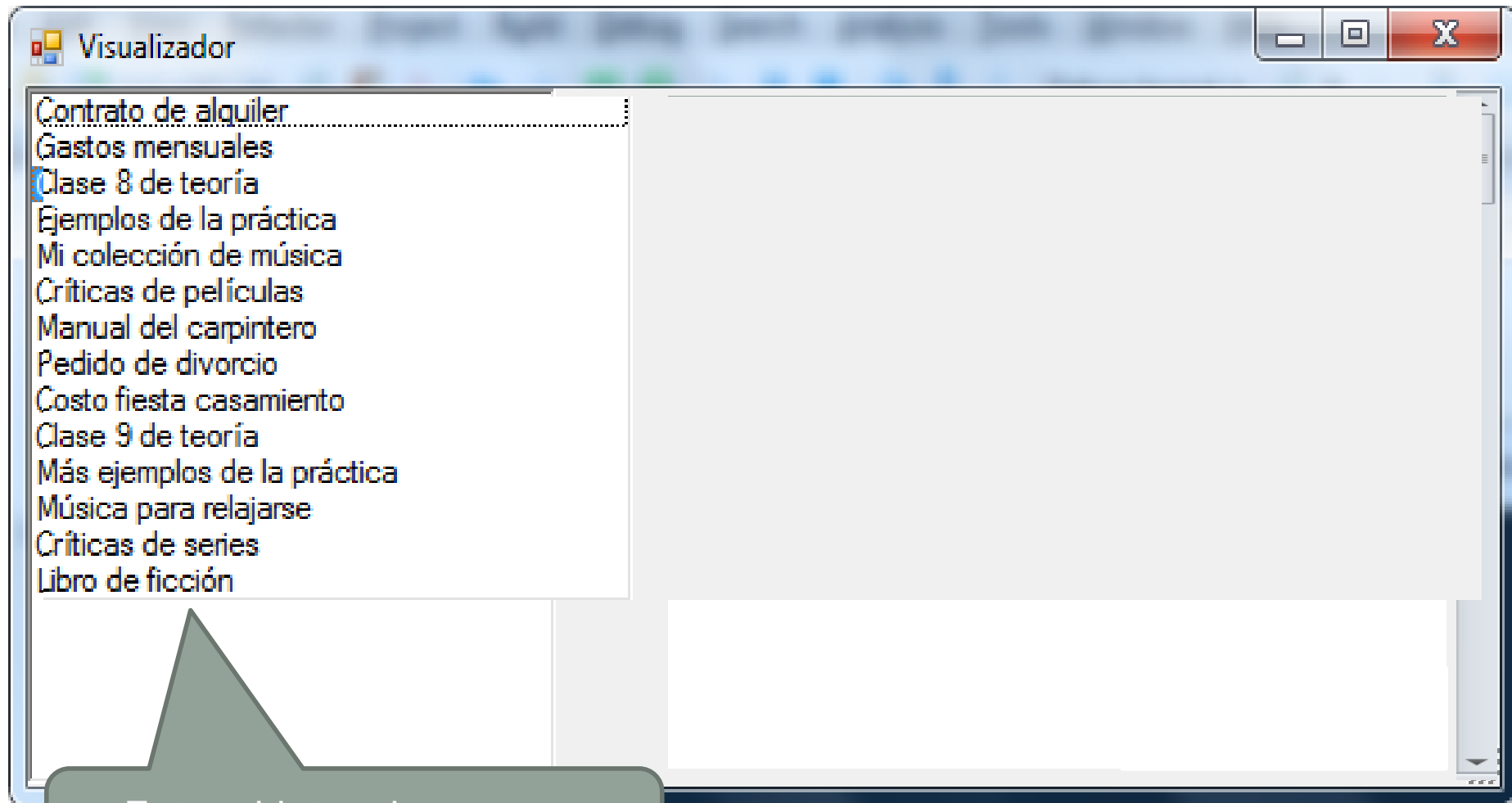
    void mostrarVistaPrevia()
        if(documentoReal == null)
            documentoReal = new DocumentoDeTexto(nombre)
```

```
documentoReal.mostrarVistaPrevia()
```



Luego reenviamos la  
petición al documento real

# Proxy - Ejemplo



Estos objetos siempre son  
proxies



# Proxy - Implementación

```
class DocumentoDeTextoProxy : IDocumento
    . . .

    void mostrarVistaPrevia()
        if(documentoReal == null)
            documentoReal = new DocumentoDeTexto(nombre)

        documentoReal.mostrarVistaPrevia()
```

El proxy actúa como tal todo lo posible para retrasar la construcción del sujeto real, luego redirecciona todas las peticiones una vez que éste fue creado.

# Proxy - Implementación

```
class Aplicacion
    void iniciarAplicacion(archivos)
        foreach a in archivos:
            lista.Add(fabrica.crearDocumento(a))
```

```
class FabricaDeDocumentos
    . . . new DocumentoDeTextoProxy()
    . . . new PlanillaDeCalculoProxy()
    . . . new PresentacionConDiapositivasProxy()
    . . . New ImagenProxy(new ImagenAdapter(new Imagen()))
```

# Proxy – Ventajas

- Un proxy puede ocultar al objeto real, creando objetos de manera eficiente solo por encargo
- Permite hacer tareas de mantenimiento y optimización extras.
- Un proxy puede actuar como contador de acceso, contabilizando todas las veces que se invoca un determinado método.