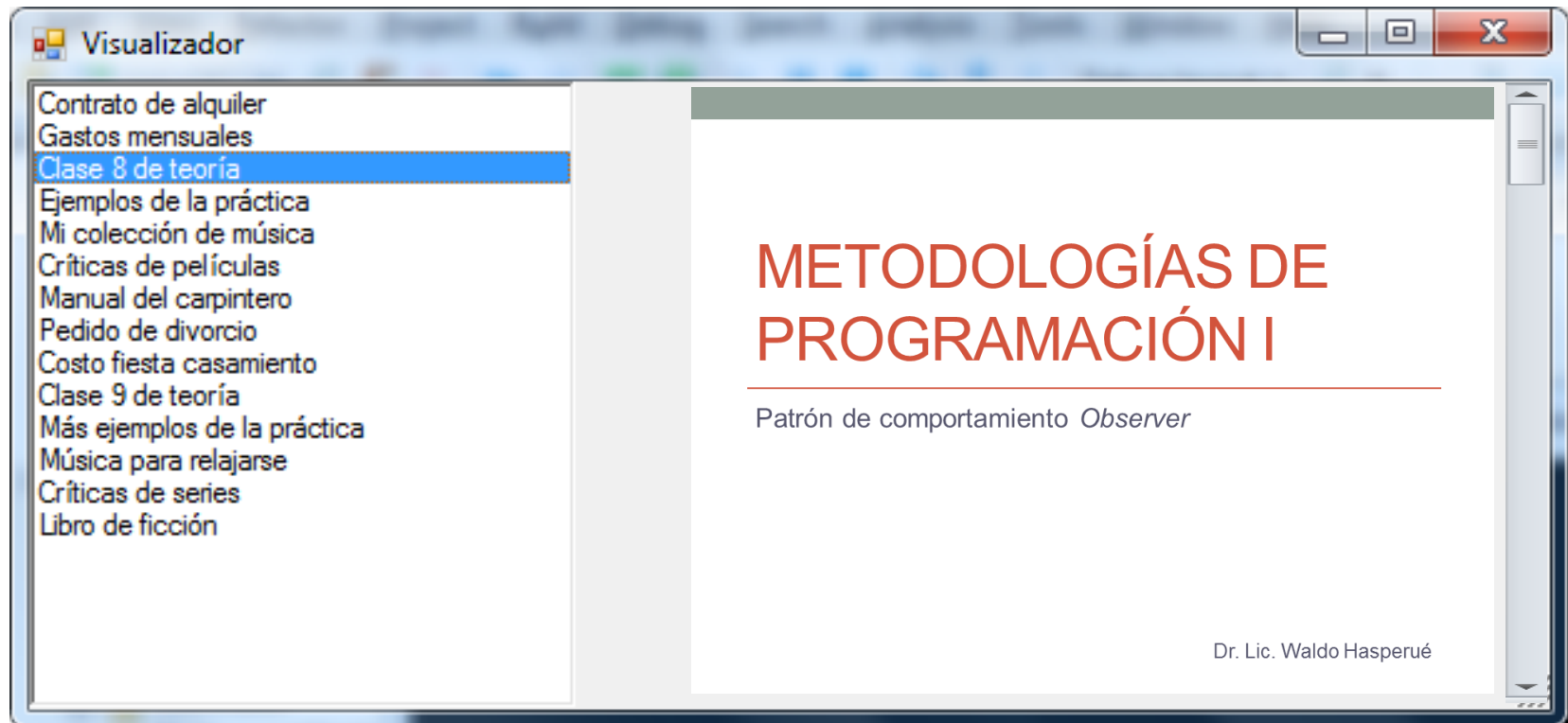


METODOLOGÍAS DE PROGRAMACIÓN I

Patrón estructural *Adapter*

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

- Nuestra aplicación es capaz de visualizar documentos de texto, presentación de diapositivas, planillas de cálculo y álbumes de música.
- Nos solicitan que la aplicación pueda visualizar imágenes.
- ¿Qué debemos hacer? Una nueva clase que implemente *IDocumento* que sea capaz de mostrar vistas previas de imágenes:

```
class Imagen : IDocumento
    string getNombre()
        . . .
    void mostrarVistaPrevia()
        . . .
```

Situación de ejemplo

- Por suerte conseguimos una clase que hace esa tarea por lo que no tenemos que implementar nada.
- El único problema es que esa clase implementa una interface *IImage*.

```
interface IImage
    string getName()
    void showPreview()
```

- La cual no coincide con nuestra interface *IDocumento*.

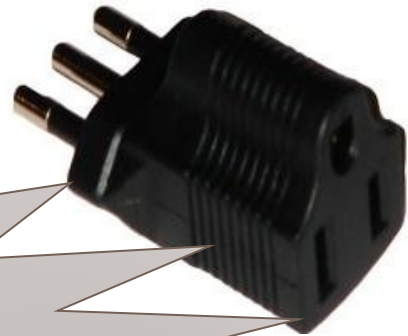
Problema

¿Qué podemos hacer?

- Modificar la interface *Image* para que sea compatible con nuestras interfaces...
- Modificar todo nuestro código para acomodarlo a la interface *Image*...
- Muchas veces ninguna de estas dos alternativas es viable, por el costo que representa. Los grandes sistemas poseen cientos y cientos de clases y las interfaces poseen decenas de métodos. Modificarlos a todos no es trivial.
- Además, algunas clases son reutilizadas por muchos sistemas, no podemos cambiar todo.

Motivación

¿Es posible contar con un mecanismo que permita usar más de una interface como si fuera la misma?



El patrón Adapter
permite resolver
este problema

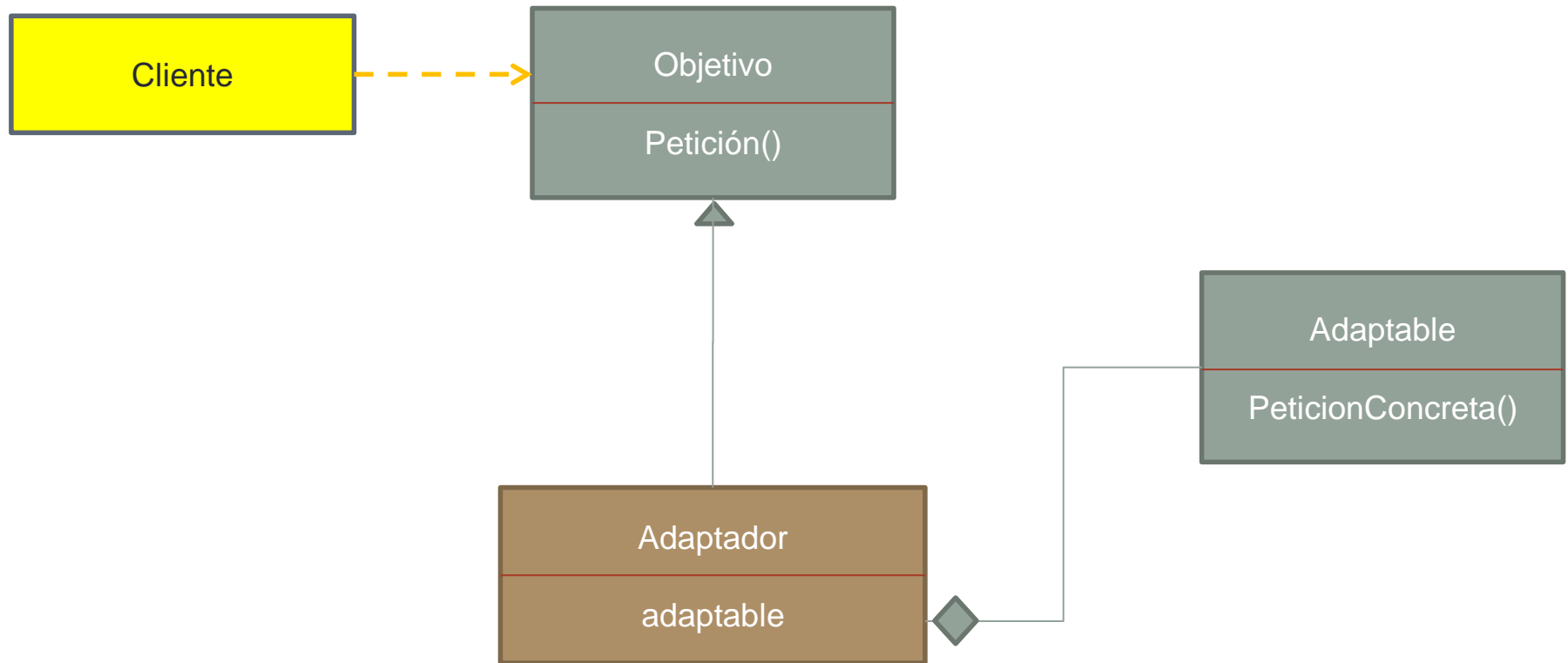
Adapter

Propósito: Convierte la interface de una clase en otra interface que es la que esperan los clientes. Permite que cooperen clases que de otra forma no podrían por tener interfaces incompatibles.

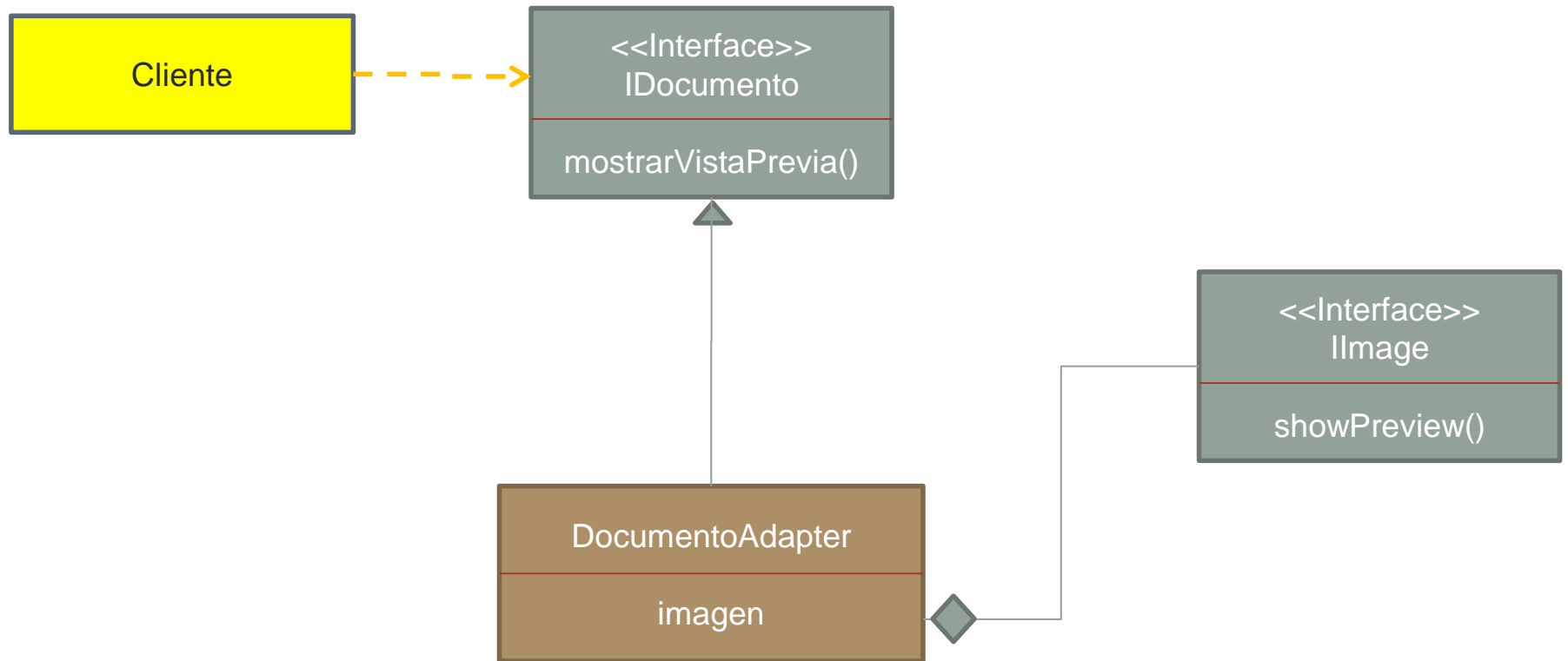
Aplicabilidad: usarlo cuando

- Se quiere usar una clase existente y su interface no concuerda con la que necesita.
- Se quiere crear una clase reutilizable que coopere con clases no relacionadas o que no han sido previstas.
- Es necesario usar varias subclases existentes, pero no resulta práctico adaptar su interface heredando de cada una de ellas.

Adapter - Estructura



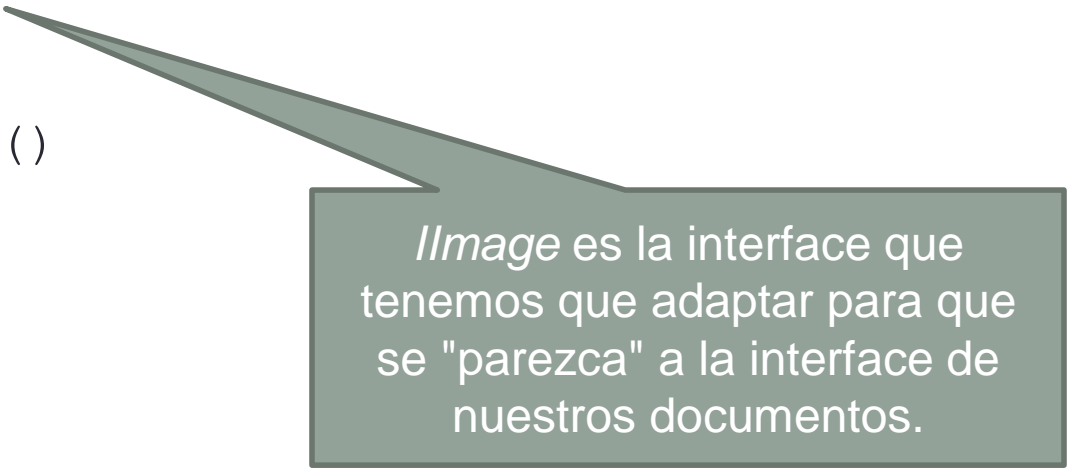
Adapter - Estructura



Adapter - Implementación

```
interface IImage
    string getName()
    void showPreview()
```

```
interface IDocumento
    string getNombre()
    void mostrarVistaPrevia()
```



IImage es la interface que tenemos que adaptar para que se "parezca" a la interface de nuestros documentos.

Adapter - Implementación

```
class DocumentoAdapter : IDocumento
    IImage imagen

    constructor (IImage i)
        imagen = i

    string getNombre()
        return imagen.getName()

    void mostrarVistaPrevia()
        imagen.showPreview()
```

Adapter - Implementación

```
class DocumentoAdapter : IDocumento
```

```
    IImage imagen
```

```
    constructor (IImage i)
```

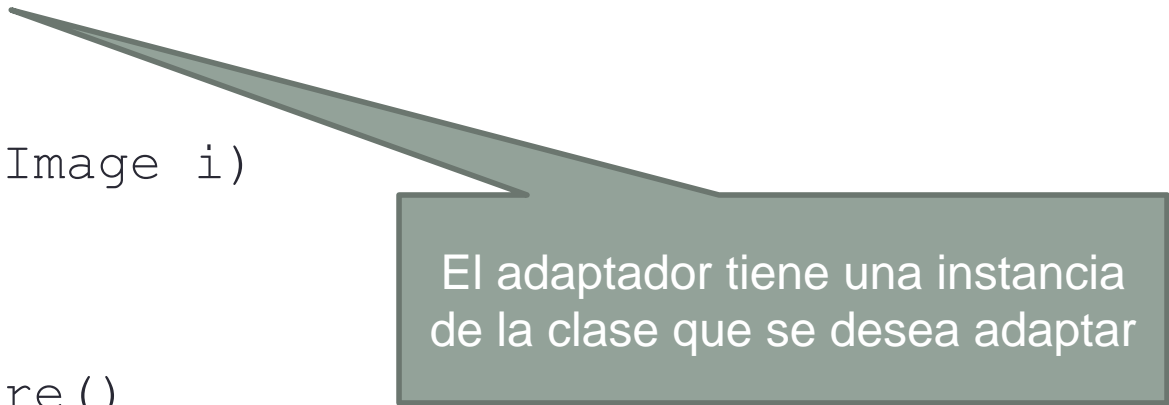
```
        imagen = i
```

```
    string getNombre()
```

```
        return imagen.getName()
```

```
    void mostrarVistaPrevia()
```

```
        imagen.showPreview()
```



El adaptador tiene una instancia de la clase que se desea adaptar

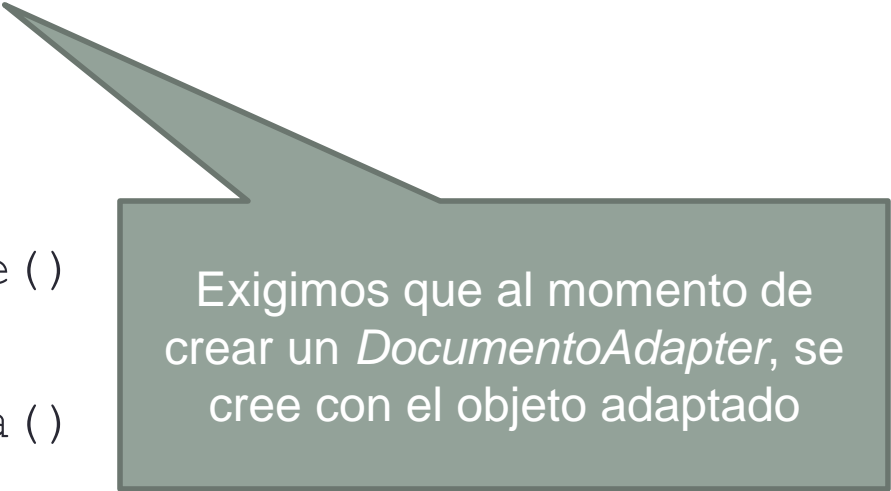
Adapter - Implementación

```
class DocumentoAdapter : IDocumento
    IImage imagen

    constructor (IImage i)
        imagen = i

    string getNombre()
        return imagen.getName()

    void mostrarVistaPrevia()
        imagen.showPreview()
```



Exigimos que al momento de crear un *DocumentoAdapter*, se cree con el objeto adaptado

Adapter - Implementación

```
class DocumentoAdapter : IDocumento
    IImage imagen

    constructor (IImage i)
        imagen = i

    string getNombre()
        return imagen.getName()

    void mostrarVistaPrevia()
        imagen.showPreview()
```



El objeto *DocumentoAdapter* "traduce" la interface *IDocumento* a la interface propia de *IImage*.

Adapter - Implementación

```
metodo()
```

```
List lista
```

```
imagenAdaptada = new Imagen("Ejemplo")
```

```
docuAdaptador = new DocumentoAdapter(imagenAdaptada)
```

```
lista.Add(docuAdaptador)
```

Para poder usar la clase *Imagen* hay que instanciar un *DocumentoAdapter*.

Adapter – Ventajas

- Adopta una clase Adaptable a Objetivo, pero se refiere únicamente a una clase Adaptable concreta.
- Permite que Adaptador redefina parte del comportamiento de Adaptable, por ser Adaptador una subclase de Adaptable.
- Introduce un solo objeto para obtener el objeto adaptado.
- Permite que un Adaptador funcione con muchos adaptables (si este es una superclase)