

11

# Adapter

## Diseño del Software

Grado en Ingeniería Informática del Software

2024-2025

# **Patrón estructural de clases y objetos**

*Convierte la interfaz de una clase  
en otra que es la que esperan los  
clientes. Permite que trabajen  
juntas clases que de otro modo no  
podrían por tener interfaces  
incompatibles.*

**También conocido  
como  
Wrapper (Envoltorio)**

# Motivación

Consideremos, por ejemplo, un editor de dibujos que permite a los usuarios dibujar elementos gráficos como líneas, polígonos, texto, etcétera.

La abstracción clave del editor de dibujo es el objeto gráfico **Shape**, que puede dibujarse a sí mismo.

El editor define una subclase de Shape para cada tipo de objeto gráfico: una clase **LineShape** para líneas, una clase **PolygonShape** para polígonos, etcétera.

**Text**



Una clase `TextShape` es bastante más difícil de implementar, por lo que decidimos utilizar una clase `TextView` proporcionada por una biblioteca externa.

Como es natural, `TextView` no es del tipo `TextShape`, así que ¿cómo podemos hacer que funcione con nuestro editor?

**¿Cómo pueden funcionar clases existentes y no relacionadas en una aplicación que espera clases con una interfaz diferente?**

¿Cambiaremos la clase TextView  
para que se ajuste a la interfaz  
Shape?

¿Cambiamos la clase TextView  
para que se ajuste a la interfaz  
Shape?

Lo anterior no es posible a menos que  
dispongamos del código fuente de la  
biblioteca externa.

¿Cambiamos la clase TextView para que se ajuste a la interfaz Shape?

Lo anterior no es posible a menos que dispongamos del código fuente de la biblioteca externa.

Pero, incluso en ese caso, ¿tendría sentido cambiar el código de la biblioteca para tratar directamente con los tipos de una aplicación concreta?

Clase adaptada

**TextView**

+getExtent()

**TextShape**

-text: TextView

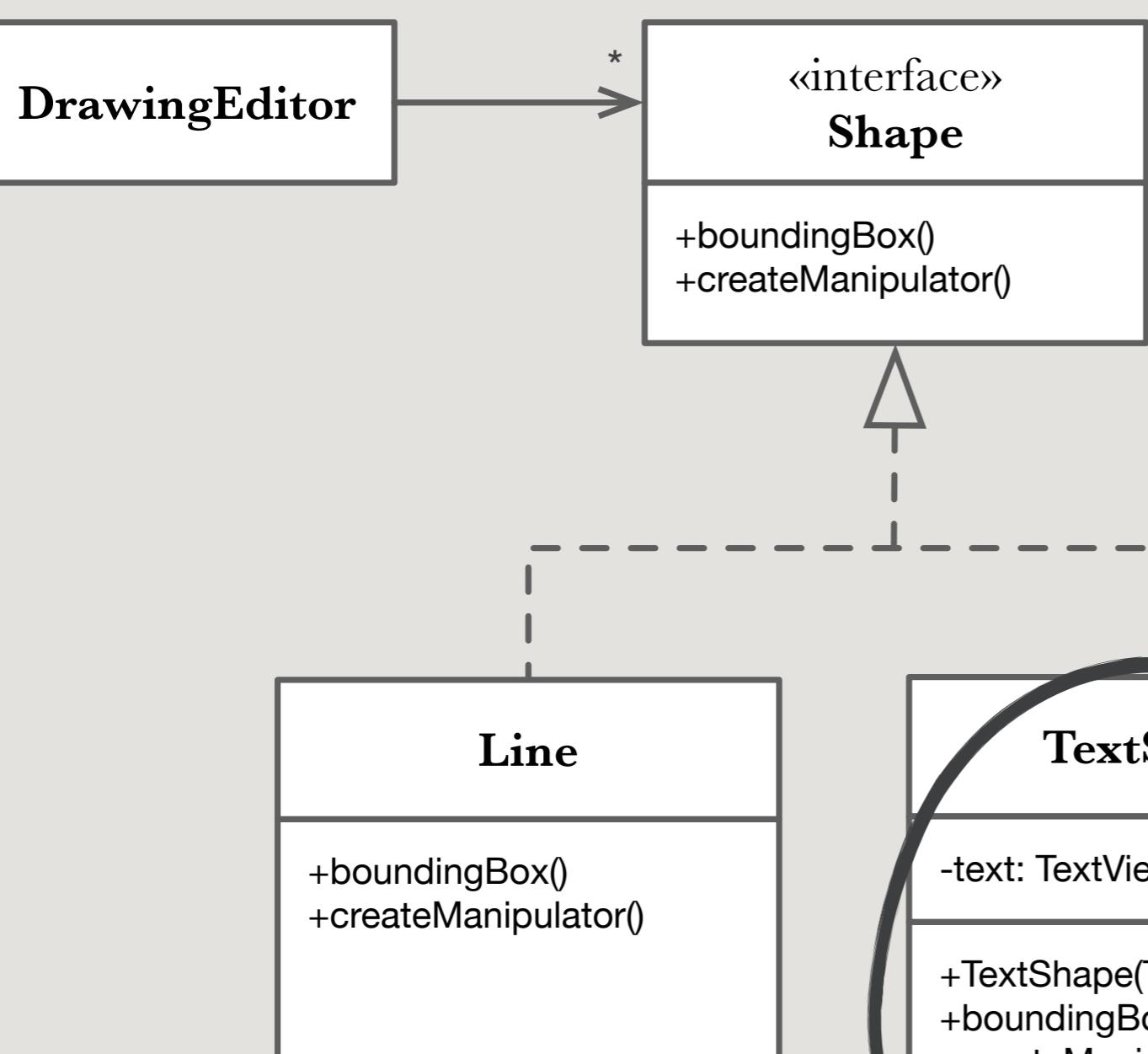
+TextShape(TextView)  
+boundingBox()  
+createManipulator()

Adaptador

text

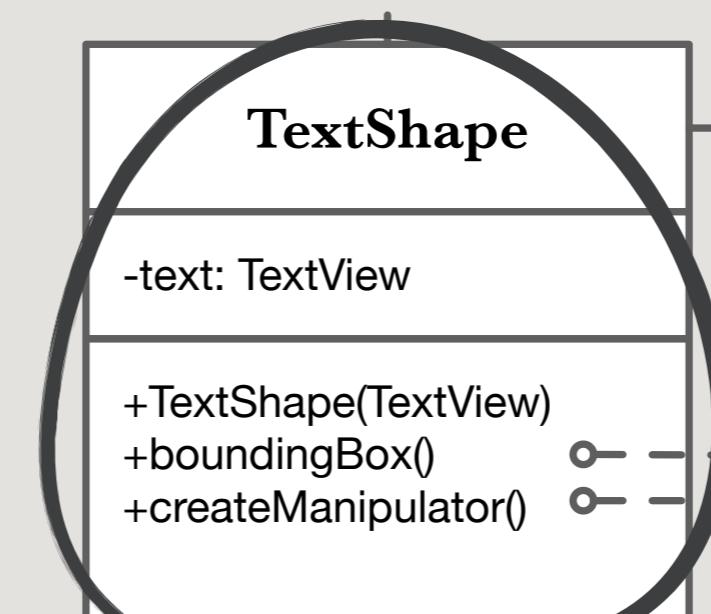
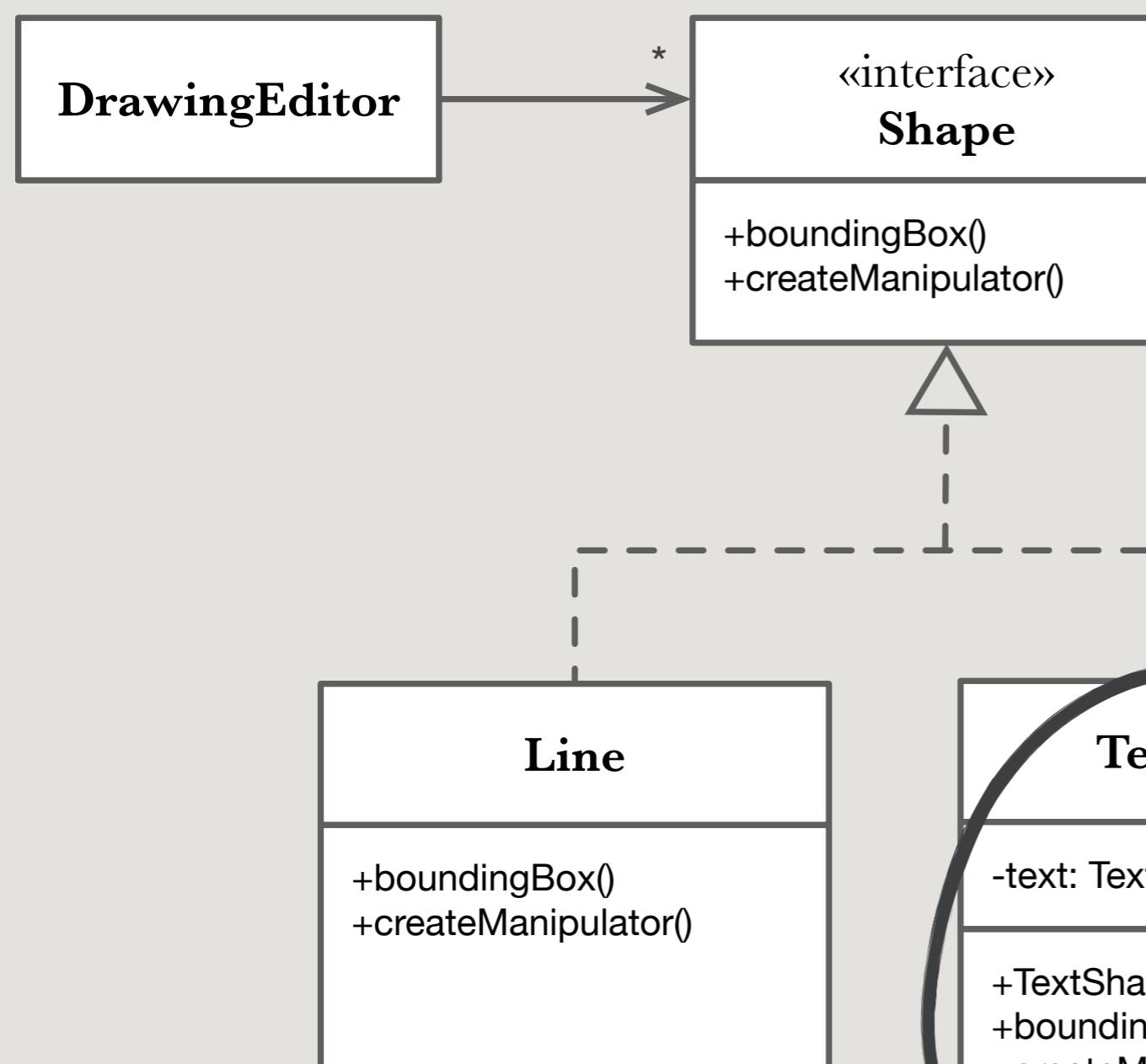
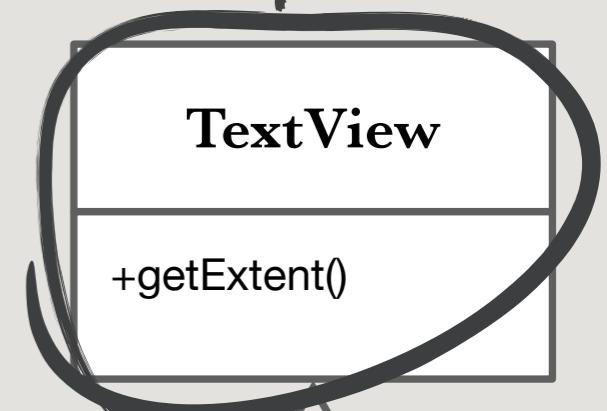
return text.getExtent();

return new TextManipulator();



La versión de objetos del patrón Adaptador aplicada al problema anterior

Clase adaptada



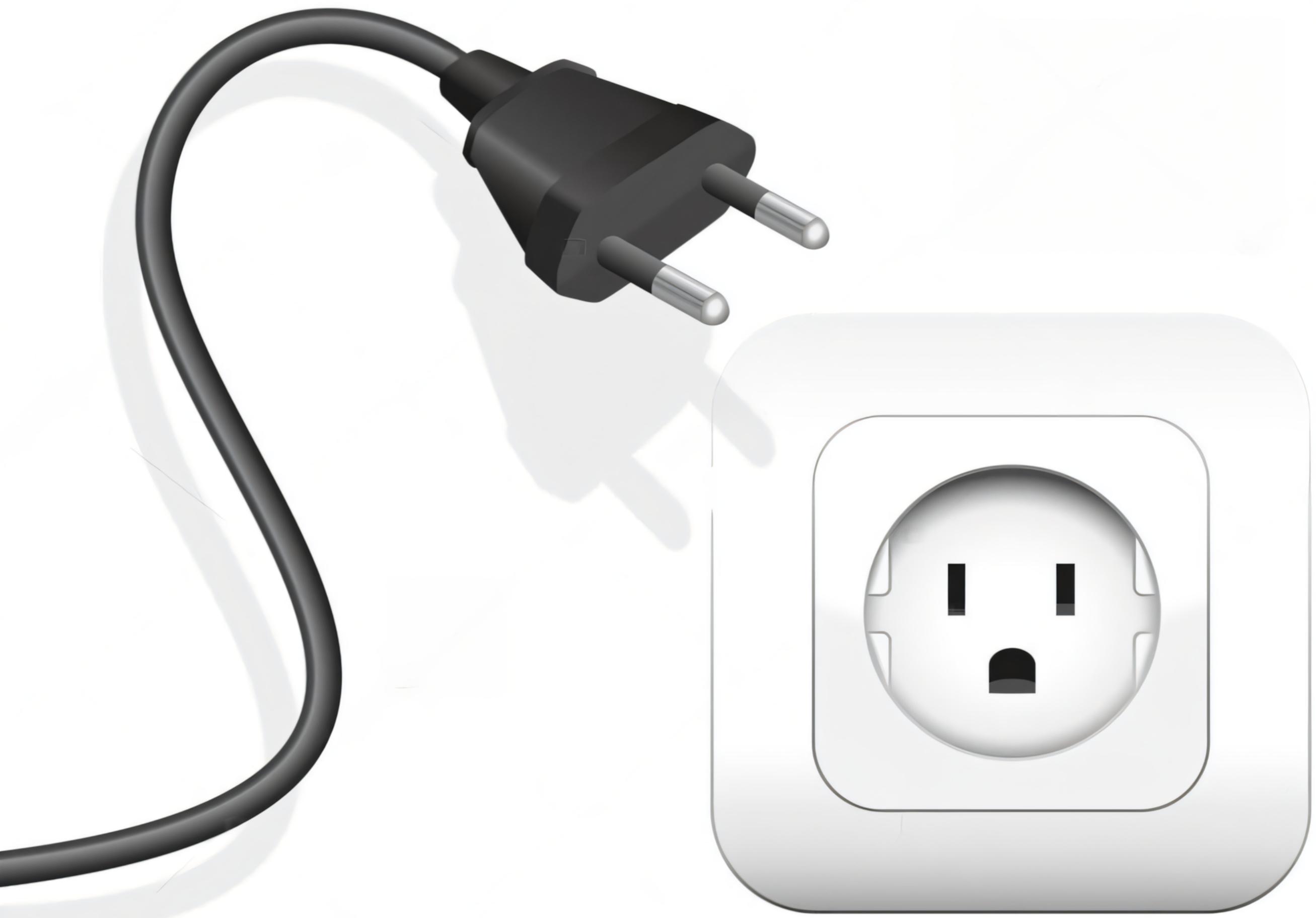
Adaptador

text

```
return getExtent();
```

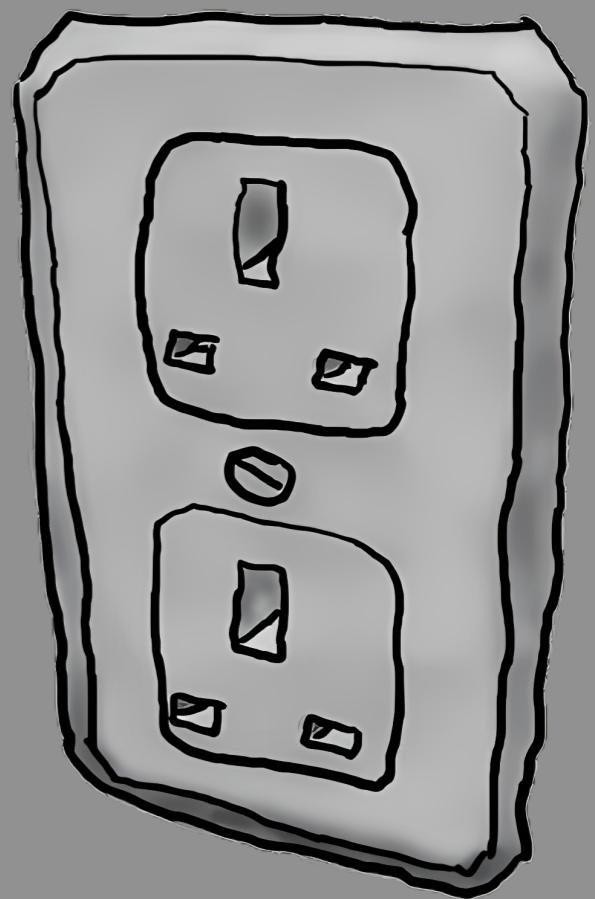
```
return new TextManipulator();
```

La versión de clases del patrón Adaptador aplicada al problema anterior

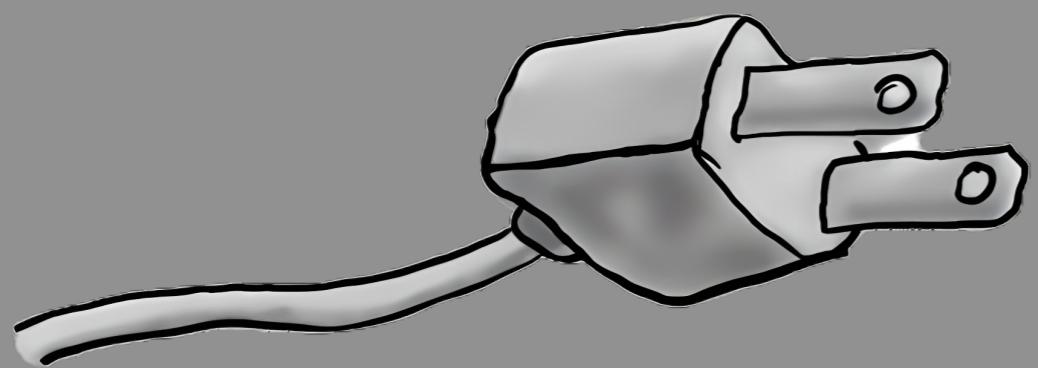




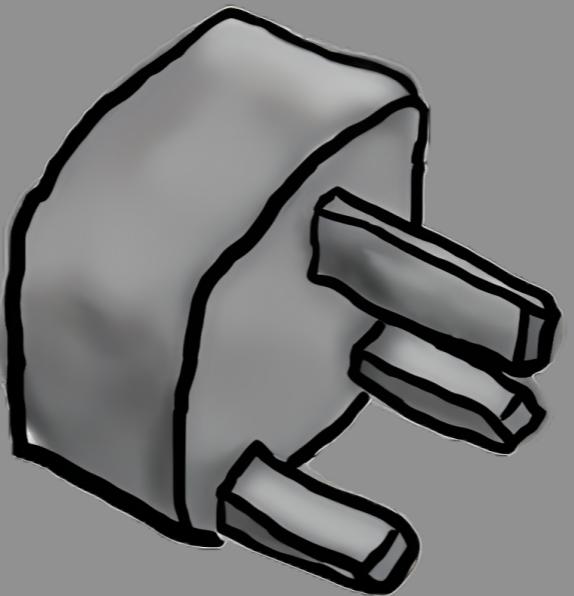
Target



Adaptee



Adapter





9:41



Ver

Conversación



Hello

Hola

Buenos días

Good morning

How are you?

¿Cómo estás?

Estoy bien, gracias!

I'm fine, thank you!

Inglés (EE. UU.)

Enter text



Traducción



Cámara



Conversación



Favoritas

# Aplicabilidad

Úsese el patrón Adapter cuando

**Queremos utilizar una clase  
existente y su interfaz no  
coincide con la que necesitamos.**

**Queremos crear una clase  
reutilizable que coopere con otras  
con las que no está relacionada.**

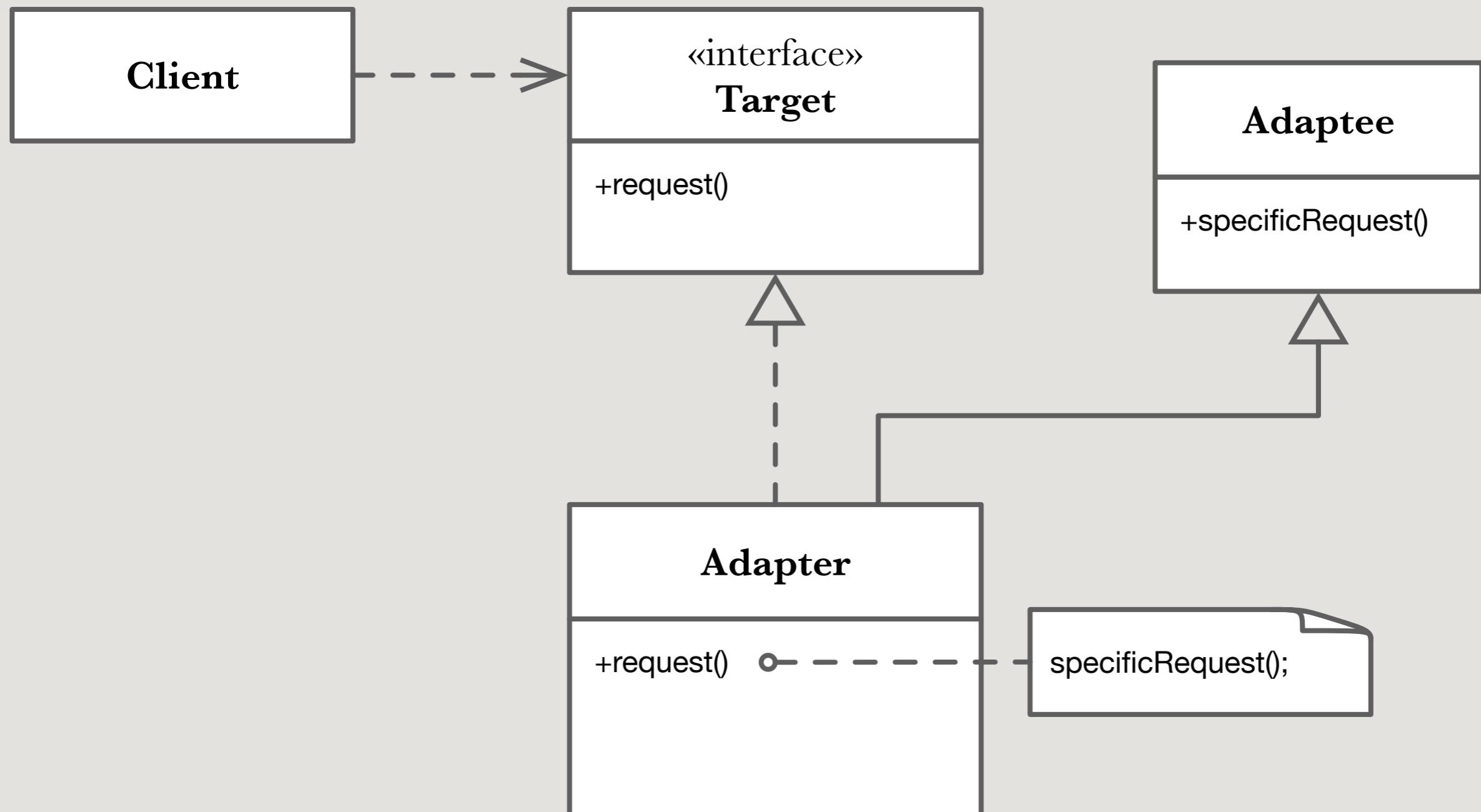
Que no tendrán, por tanto, interfaces  
compatibles.

(Solo la versión de objetos)

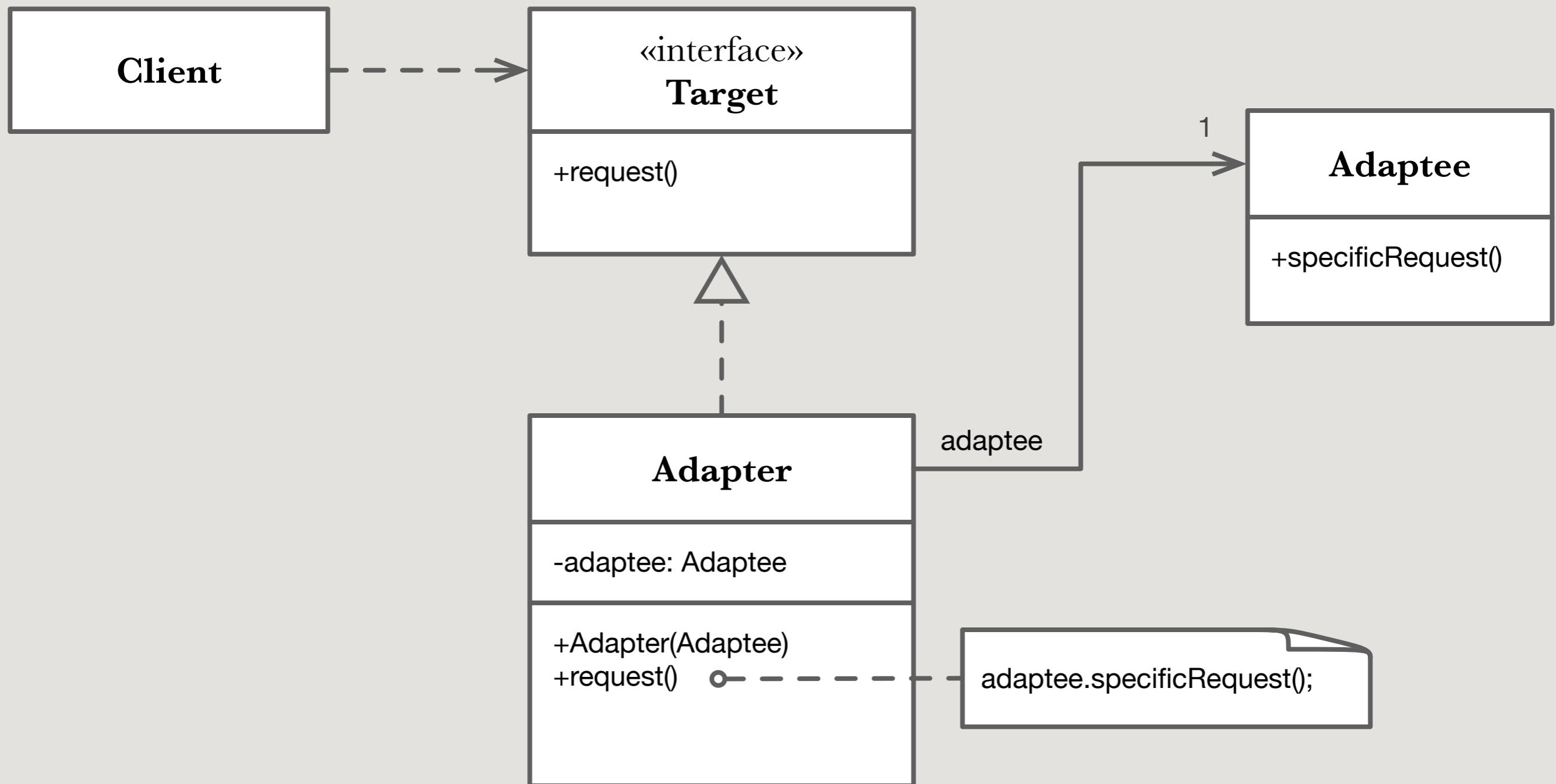
**Necesitamos usar varias subclases existentes pero sin tener que adaptar su interfaz creando una nueva subclase de cada una.**

Un adaptador de objetos puede adaptar la interfaz de su clase padre.

# Estructura



La estructura del patrón de diseño Adapter (versión de clases)



La estructura del patrón de diseño Adapter (versión de objetos)

# Participantes

# Target

(Shape)

**Define la interfaz específica del dominio,  
que es la que utilizan los clientes.**

# Client

(DrawingEditor)

Colabora con objetos de tipo Target.

# Adaptee

(TextView)

La clase existente cuya interfaz necesita ser adaptada.

# Adapter

(TextShape)

**Adapta la interfaz de Adaptee a la de Target.**

# Colaboraciones

Los clientes llaman a las operaciones del adaptador.

A su vez, el adaptador llama a las operaciones del objeto adaptado que llevan a cabo la petición.

# Consecuencias

# Diferencias entre el adaptador de clases y de objetos

# **Un adaptador de clases adapta una clase adaptadora concreta.**

Por tanto, no funcionará cuando queramos adaptar una clase y todas sus subclases.

**Un adaptador de objetos**, por el contrario, adapta el propio adaptador y todas sus subclases.

# **Un adaptador de clases es del mismo tipo que el adaptado.**

Puede ser pasado a cualquier objeto que espere un adaptador. Es transparente para los clientes del adaptador.

Eso no ocurre con el adaptador de objetos (no se puede utilizar allí donde puede hacerlo un objeto adaptador).

¿Cuánta  
adaptación?

**Los adaptadores pueden hacer más o menos trabajo para adaptar el objeto adaptado a la interfaz esperada.**

Puede ir desde una mera conversión simple de llamadas a métodos, llamando a los mismos o parecidos del adaptado a tener que implementar un conjunto de operaciones totalmente nuevas.

Depende de lo parecidas que sean la clase que hay que adaptar y la clase objetivo.

# Otras

# **Single Responsibility Principle**

Separa el código adaptado de la lógica de negocio del programa.

# **Open/Closed Principle**

Se pueden introducir nuevos tipos de adaptadores en el programa sin cambiar el código existente.

# Implementación

# Adaptadores conectables

Es posible construir una especie de adaptación de interfaces en la propia clase, a priori.

# Patrones relacionados

# Decorator

Aunque ambos tienen una estructura similar, sus propósitos difieren: un decorador **añade responsabilidades a un objeto, sin cambiar su interfaz.**

Además, permite combinarlos de forma recursiva, algo que los adaptadores no (tampoco están pensados para ello, no obstante).