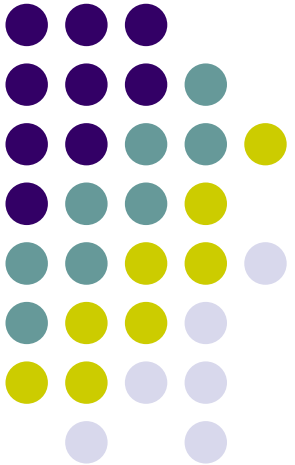
Patrones de Diseño

Patrón estructural *Adapter*



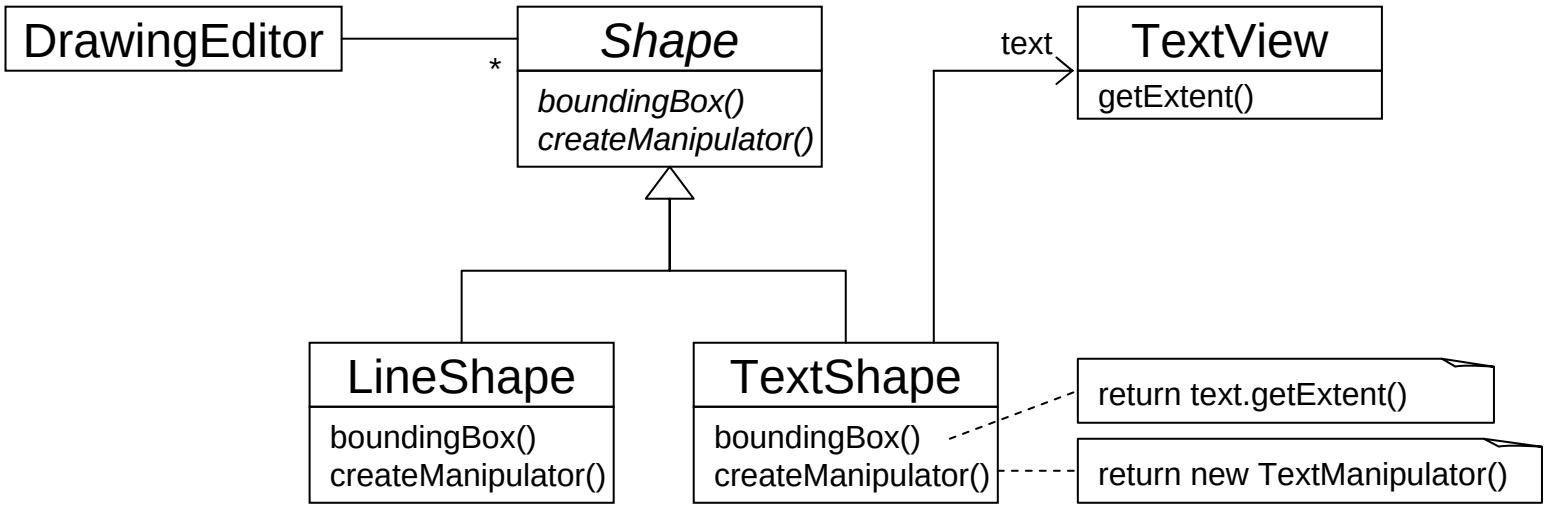
Adapter
Propósito

- Convertir la interfaz de una clase en otra distinta que espera el cliente
- Permitir que un conjunto de clases con interfaces incompatibles trabajen juntas
- También conocido como *wrapper* (envoltorio)

Adapter
Motivación

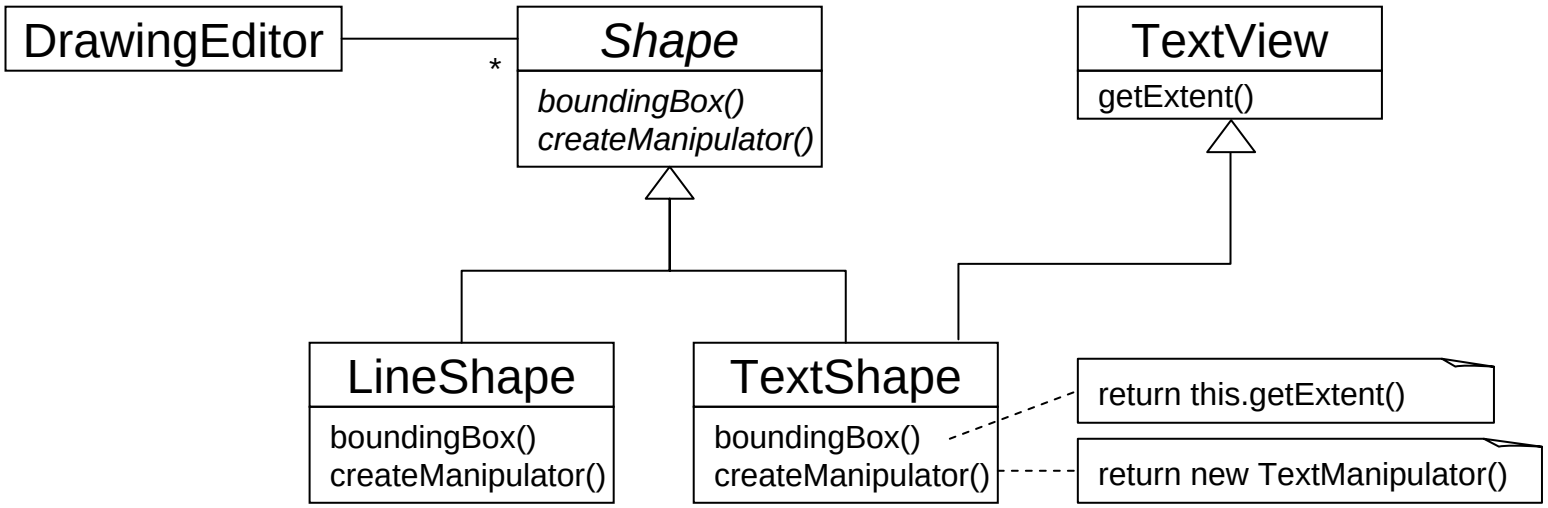
- Ej: Editor de elementos gráficos (líneas, texto, etc...)
- Clase abstracta *Shape* con interfaz de los objetos gráficos, y una subclase por cada tipo de objeto gráfico (*LineShape*, *TextShape*, ...).
- Existe clase externa *TextView* no intercambiable con *TextShape*
- Soluciones:
  - Hacer *TextView* conforme a *Shape*
    - No es posible sin el código fuente
    - No es adecuado modificar *TextView* para cada aplicación concreta
  - Hacer que *TextShape* adapte la interfaz de *TextView* a la de *Shape*
    - Delegación (*object adapter*)
    - Herencia múltiple (*class adapter*)

Adapter
Motivación



(variante *object adapter*)

Adapter
Motivación



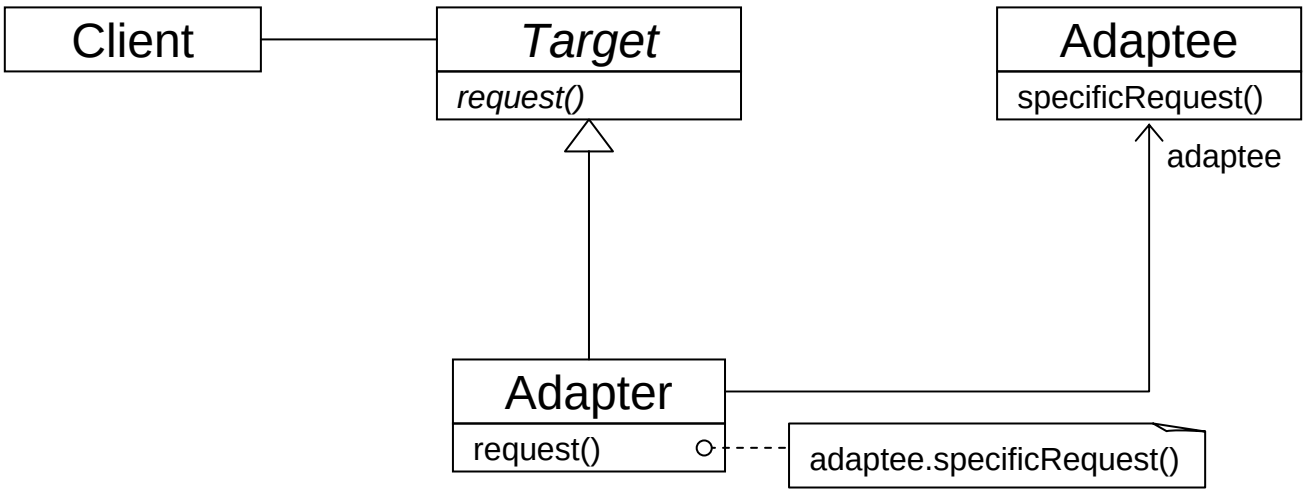
(variante *class adapter*)

Adapter
Aplicabilidad

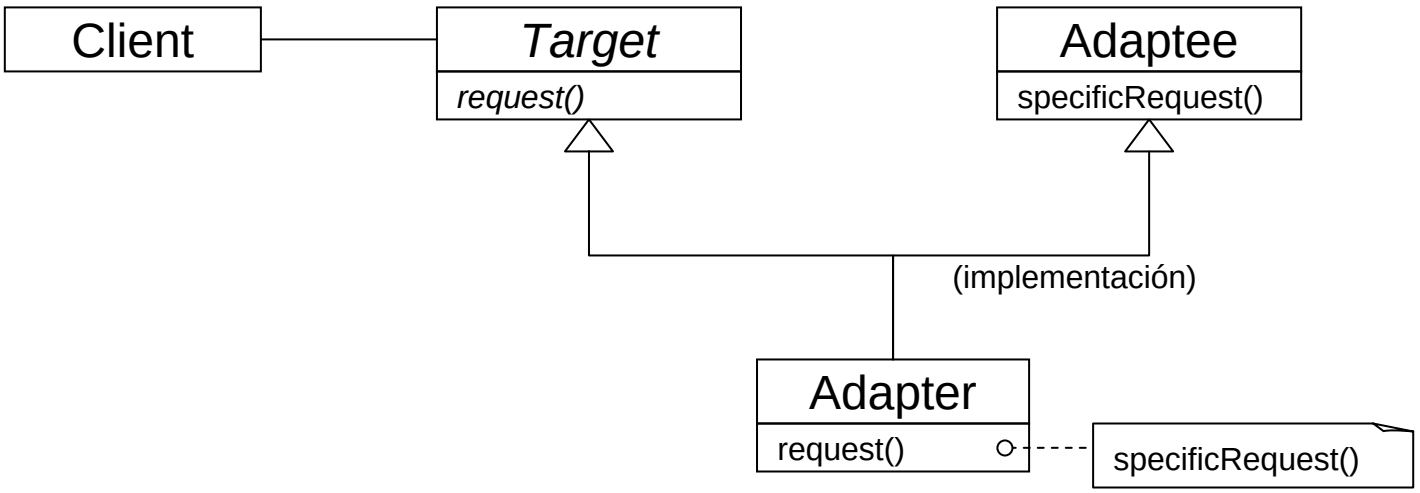
- Usa el patrón *Adapter* cuando:
  - Quieras utilizar una clase ya existente, pero cuya interfaz no coincide con la que necesitas
  - Quieras crear una clase reutilizable que coopere con otras no relacionadas, es decir, con clases que puedan no tener una interfaz compatible
  - (sólo *object adapter*) Necesites usar varias subclases existentes pero no sea práctico adaptar sus interfaces. Un *object adapter* puede adaptar la interfaz de la clase padre.

Adapter

Estructura – *object adapter*



Adapter
Estructura – *class adapter*



Adapter
Participantes

- **Target** (*Shape*): define la interfaz específica de dominio que el cliente usa
- **Client** (*DrawingEditor*): colabora con los objetos que implementan la interfaz definida por el *target*
- **Adaptee** *(TextView):* define una interfaz existente que necesita adaptarse
- **Adapter** *(TextShape):* adapta la interfaz del objeto adaptado a la definida por el *target*

Adapter
Consecuencias

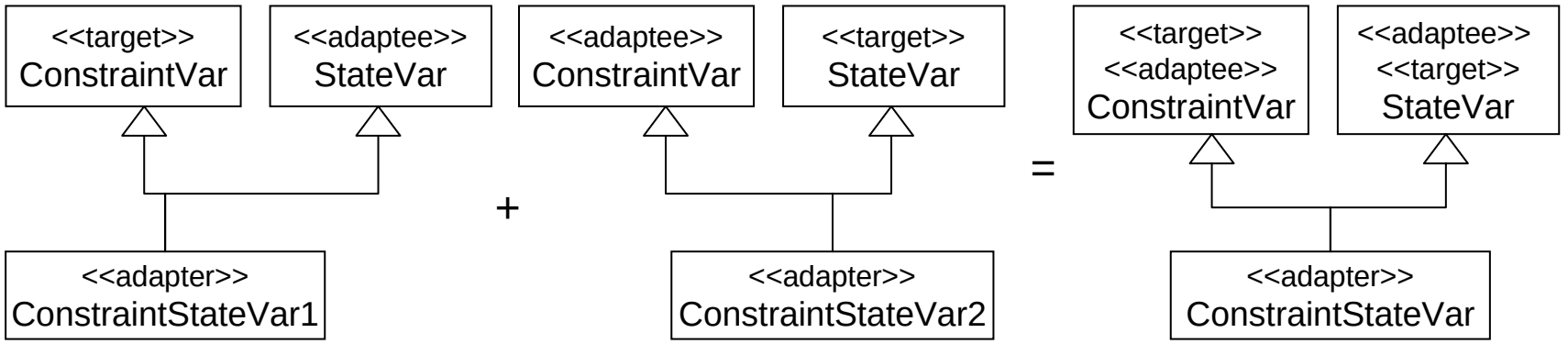
- *Object adapter*
  - Un *adapter* funciona con varios *adaptees* (el mismo *adaptee* y todas sus subclases)
  - Dificulta sobrescribir el comportamiento del *adaptee*

- *Class adapter*
  - El *adapter* hereda el comportamiento del *adaptee*, y puede sobrescribirlo
  - No sirve para adaptar una clase y todas sus subclases
  - Introduce un único objeto, no hace falta un nivel de indirección para obtener el *adaptee*

Adapter
Consecuencias

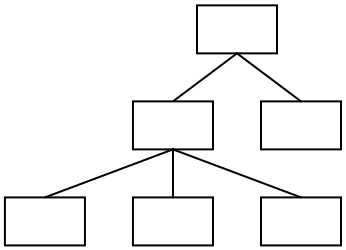
- ¿Cuánto adapta un *adapter*?
  - Depende de la similitud entre las interfaces de *adaptee* y *target*

- Adaptadores bidireccionales
  - Dos clientes necesitan ver un objeto de distinta forma



Adapter
Implementación

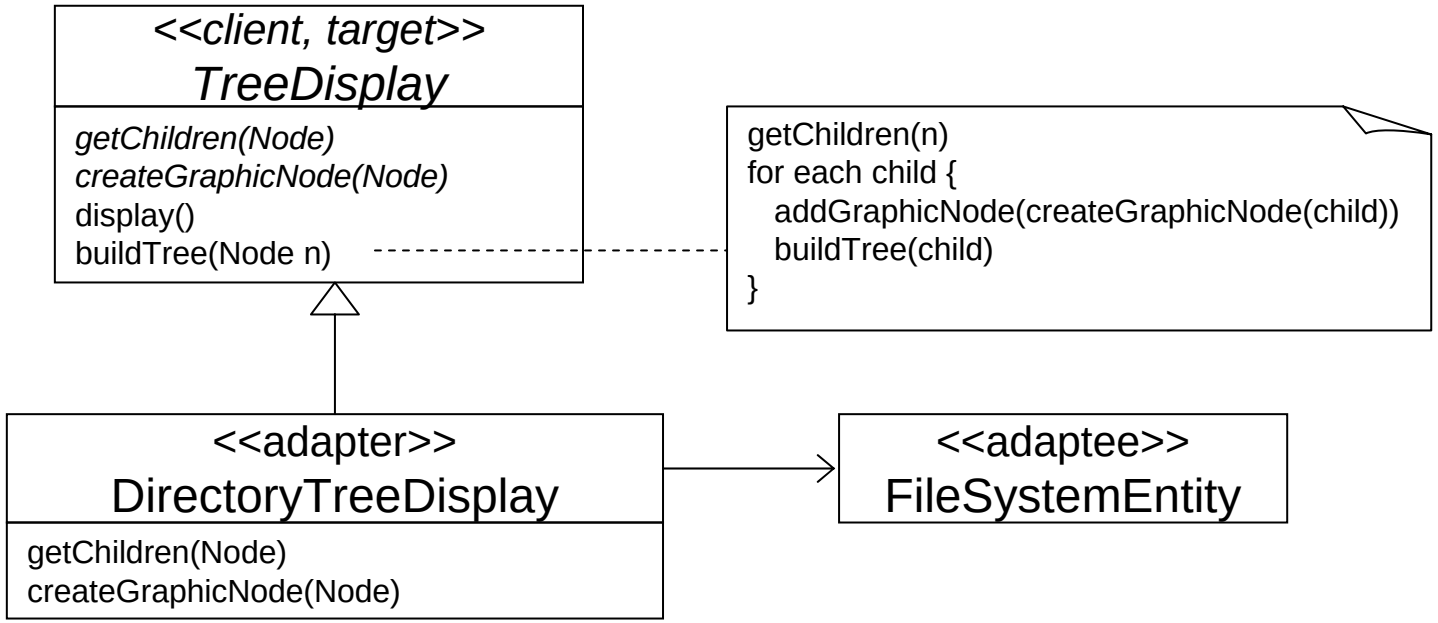
- Adaptador “conectable”
  - Maximiza la reutilización de las clases
  - Se adapta dinámicamente a una de varias clases
  - Ejemplo: construir un componente *TreeDisplay* que muestre gráficamente estructuras jerárquicas



  - Debe ser reutilizable, esto es, debe poder mostrar estructuras jerárquicas de distinto tipo, como por ejemplo:
    - jerarquía de directorios (interfaz *getSubdirectories*)
    - jerarquía de clases (interfaz *getSubclasses*)
    - etc...

Adapter
Implementación

- Adaptador “conectable”



Adapter
Ejercicio

- **Dispongo de la clase *Utilidad*, cuyo método *filtrar* elimina de un array de objetos aquellos que no cumplen cierto criterio. El método asume que los objetos del array implementan la interfaz *Filtrable*, cuyo método *cumpleCriterio* devuelve si el objeto cumple o no un criterio dado.**

- **Quiero usar la clase *Utilidad* para eliminar de un array de objetos de tipo *Registro* aquellos que son inválidos. *Registro* dispone del método *esValido* para comprobar la validez del registro, y no quiero modificar la clase para añadir nuevos métodos.**

- **¿Cómo usar *Utilidad* para filtrar los registros inválidos de un array?**

Adapter
Solución