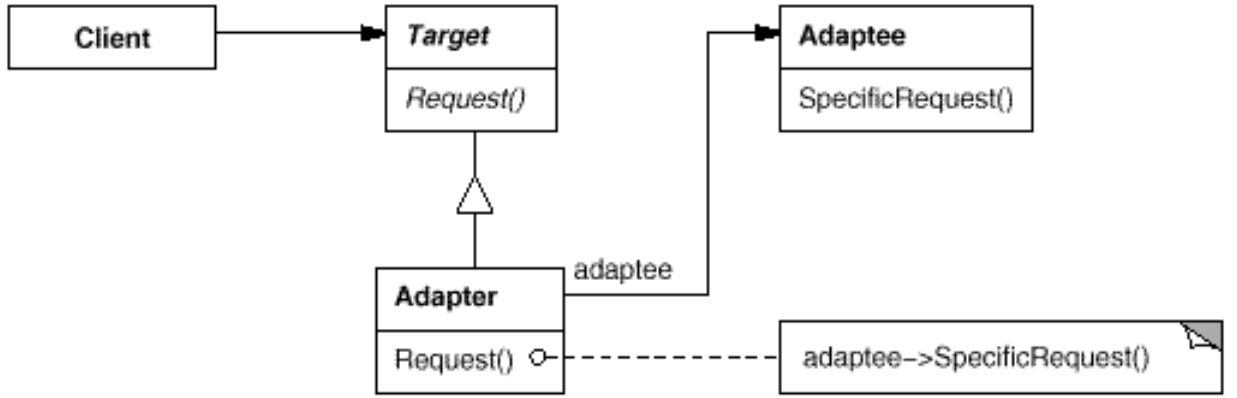
Patron de Diseño Composite

Laboratorio de Investigación y Formación en Informática Avanzada
Facultad de Informática, Universidad Nacional de La Plata
Calle 50 esq. 115 - Primer piso (1900) La Plata, Argentina
TE: (0221) 422 8252 http://www-lifia.info.unlp.edu.ar

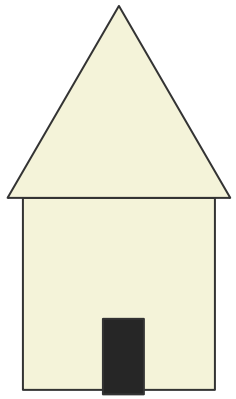
Usando Patrones. Ventajas y temas involucrados

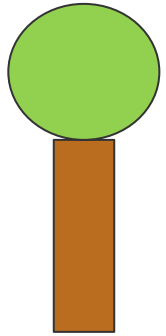
- ✓ Supongamos que conocemos patrones(e.g. Adapter).
- ✓ Conocemos problemas de diseño y buenas soluciones a ellos
- ✓ Disponemos de un vocabulario conciso y efectivo para describir problemas y soluciones
- ✓ Como mapeamos un patron a un diseño especifico?
- ✓ Como aplicamos el principio de Alexander (“use the pattern millions of times without doing the same thing twice”)?



Problema

- ✓ Supongamos que queremos manejar figuras compuestas y tratarlas como figuras simples (moverlas, rotarlas, etc).
- ✓ Y queremos tener la posibilidad de dibujar una Casa un Árbol o muchas Propiedades





Casa= Cuerpo + Techo
Cuerpo= Puerta + Pared

Arbol= Tronco + Copa

Propiedad= Casa + Arbol

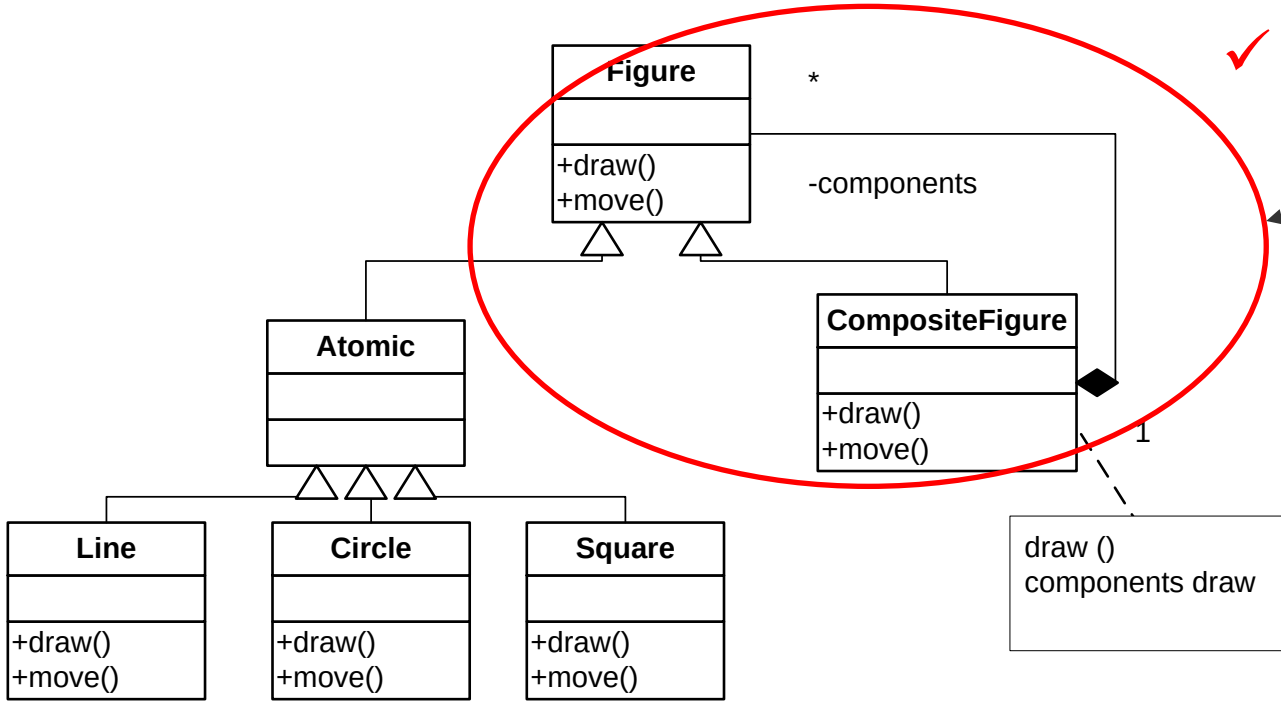
- ✓ Fundamentalmente nos interesa que para el editor no haya diferencias cuando las manipula. O sea que pueda seguir usando el protocolo de Figura

Soluciones?

- ✓ Podemos tener un arreglo de figuras y marcarlas como partes de otras….
- ✓ Como seria tal arreglo cuando hay composiciones muy “profundas”?
- ✓ Por ejemplo “Barrio” que es composición de Propiedades que a su vez es colección de Casa + Arboles....etc

Una solucion mas modular

- ✓ Tratarlas uniformemente, igual que a las simples



- ✓ Como se interpreta esta relación de conocimiento 1 a n?
- ✓ Como funciona este código?

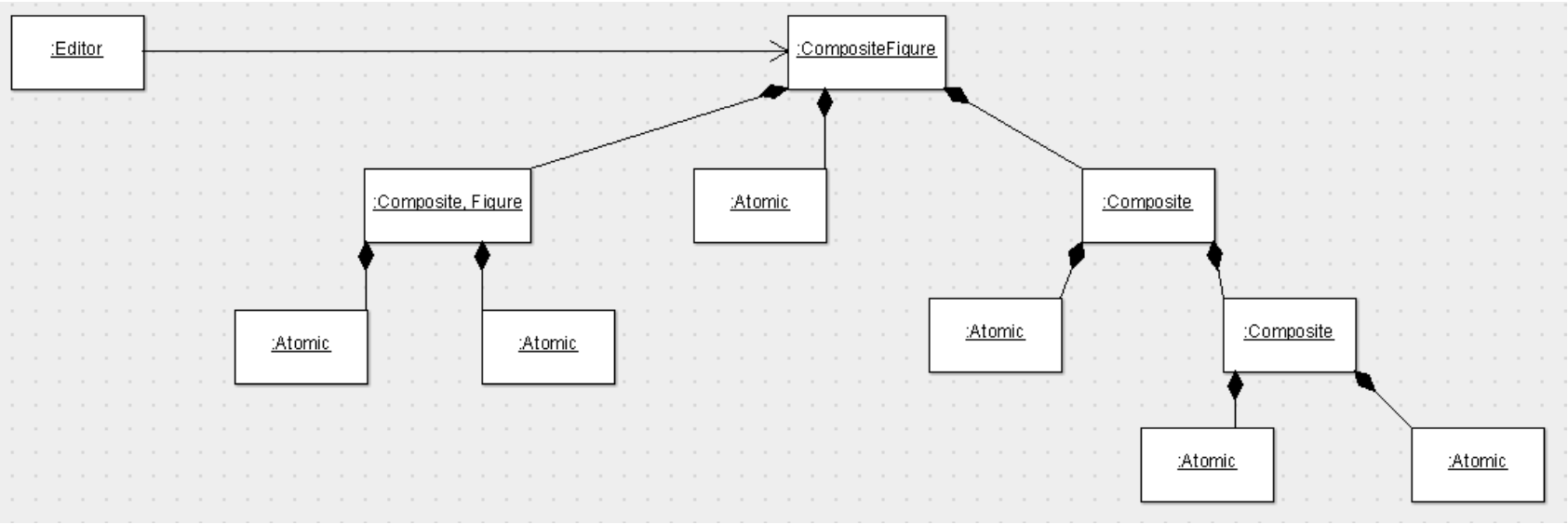
(interpretese como: para cada x en components, x draw)

- ✓ Obsérvese que un código muy simple implementa una funcionalidad que parece recursiva

Como funciona?

- ✓ El editor solo maneja figuras
- ✓ Mantenemos la interfaz polimórfica

- ✓ Así se vería una instancia de CompositeFigure



- ✓ Observen como seria el flujo de mensajes entre las “partes” de la figura

Pattern Composite

✓ Intent

Componer objetos en estructuras de arbol para representar jerarquias parte-todo. El Composite permite que los clientes traten a los objetos atomicos y a sus composiciones uniformemente

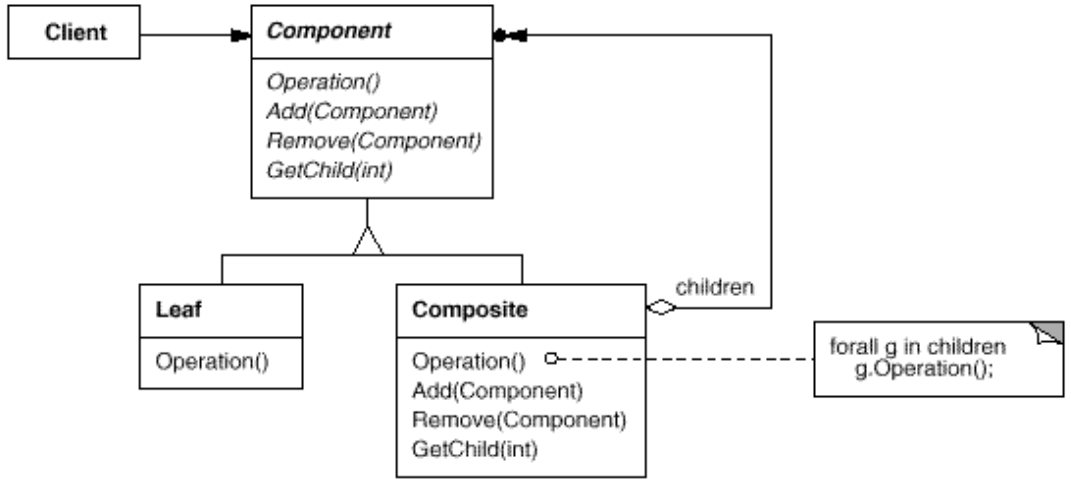
✓ Applicability

Use el patron Composite cuando

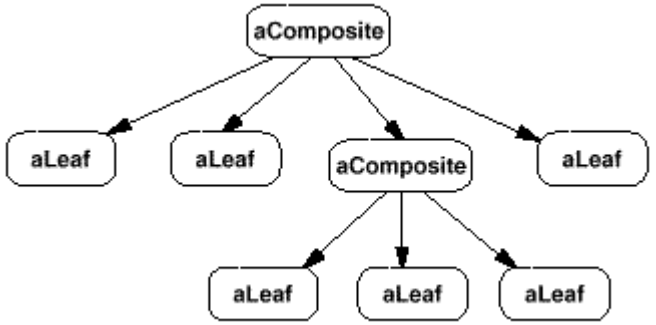
- ✓ quiere representar jerarquias parte-todo de objetos.
- ✓ quiere que los objetos “clientes” puedan ignorar las diferencias entre composiciones y objetos individuales. Los clientes trataran a los objetos atomicos y compuestos uniformemente.

Pattern Composite

✓Structure

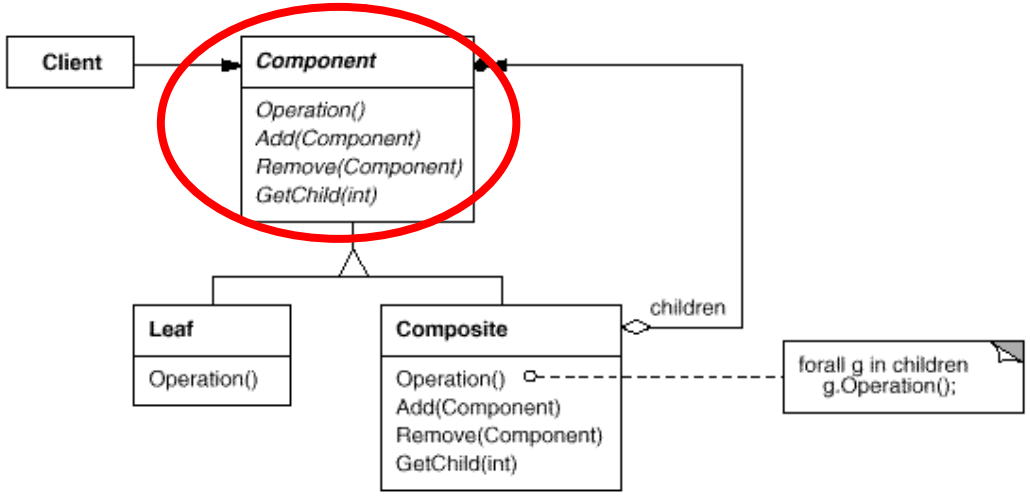


Una estructura compuesta tipica se vera asi



Composite

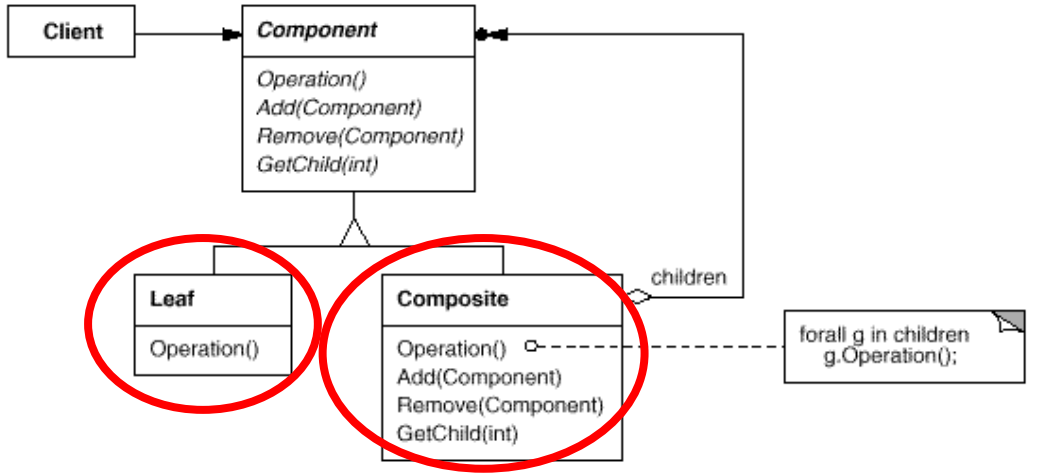
- **Participants**



- **Component** (Figure)
  - Declara la interfaz para los objetos de la composicion.
  - Implementa comportamientos default para la interfaz cc todas las clases
  - Declara la interfaz para definir y acceder “hijos”.
  - (opcional) define una interfaz para para acceder el “padre” de un componente en la estructura recursiva y la implementa si es apropiado.

Composite

- **Leaf** (Rectangle, Line, Text, etc.)
  - Representa arboles “hojas” in la composicion. Las hojas no tienen “hijos”.
  - Define el comportamiento de objetos primitivos en la composicion.



- **Composite** (CompositeFigure)
  - Define el comportamiento para componentes con “hijos”.
  - Contiene las referencias a los “hijos”.
  - Implementa operaciones para manejar “hijos”.

Pattern Composite

✓ Consequencias

El patron composite

- ✓ Define jerarquías de clases consistentes de objetos primitivos y compuestos. Los objetos primitivos pueden componerse en objetos complejos, los que a su vez pueden componerse y asi recursivamente. En cualquier lugar donde un cliente espera un objeto simple, puede aparecer un compuesto.
- ✓ Simplifica los objetos cliente. Los clientes pueden tratar estructuras compuestas y objetos individuales uniformemente. Los clientes usualmente no saben (y no deberian preocuparse) acerca de si están manejando un compuesto o un simple. Esto simplifica el código del cliente, porque evita tener que escribir código taggeado con estructura de decision sobre las clases que definen la composición

✓ Pero

- ✓ Puede hacer difícil restringir las estructuras de composición cuando hay algún tipo de conflicto (por ejemplo ciertos compuestos pueden armarse solo con cierto tipo de atómicos)