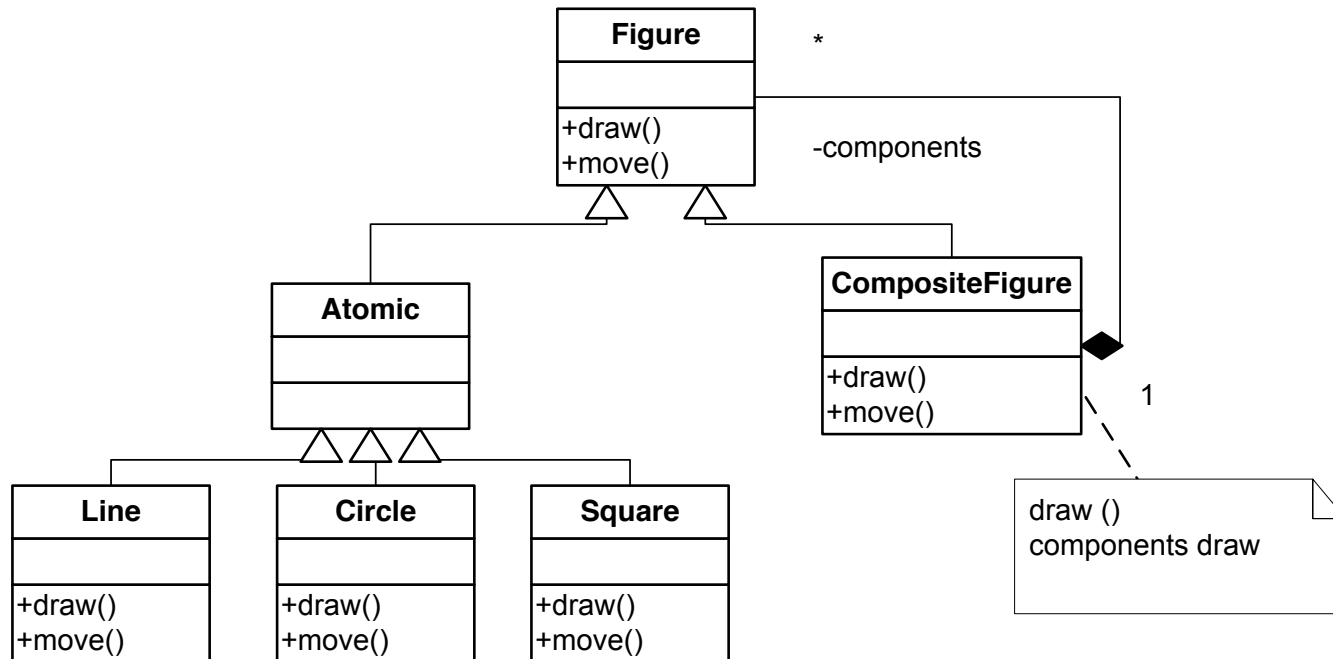


Problema

- ✓ Supongamos que queremos manejar figuras compuestas y tratarlas como figuras simples (moverlas, rotarlas, etc).
- ✓ Podemos tener un arreglo de figuras y marcarlas como partes de otras....

Una solución mas modular

- ✓ Tratarlas uniformemente



Como funciona?

- ✓ El editor solo maneja figuras
- ✓ Mantenemos la interfaz polimorfica
- ✓ Problemas? Como creamos figuras compuestas?

Una instancia del patron Composite

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

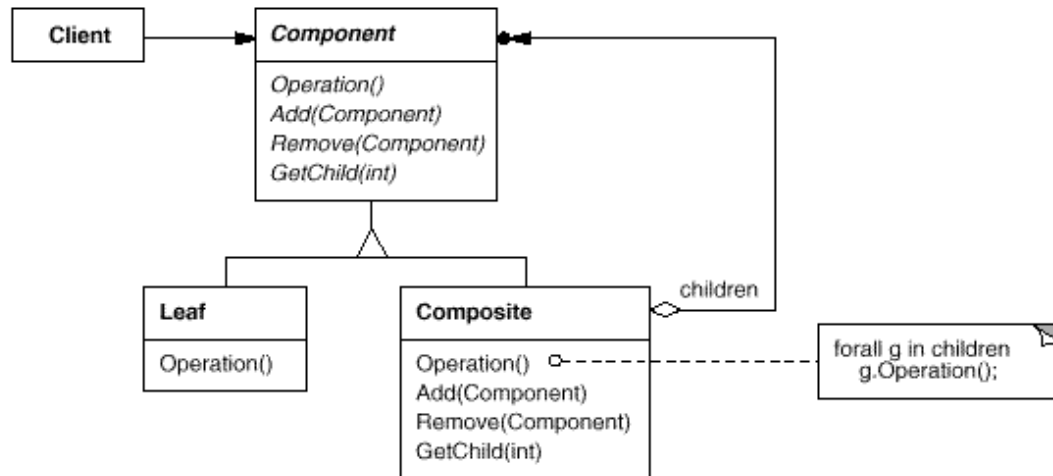
✓ Applicability

Use el patron Composite cuando

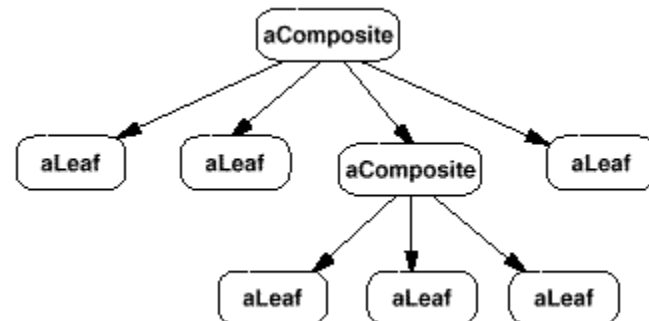
- ✓ quiere representar jerarquias parte-todo de objetos.
- ✓ quiere que los objetos “clientes” puedan ignorar las diferencias entre composiciones y objetos individuales. Los clientes trataran a los objetos atomicos y compuestos uniformemente.

Pattern Composite

✓ Structure



Una estructura compuesta
tipica se vera asi



✓ Participants

✓ Component (Figure)

- ✓ Declara la interfaz para los objetos de la composicion.
- ✓ Implementa comportamientos default para la interfaz comun a 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** (CompositePicture)
 - ✓ Define el comportamiento para componentes con “hijos”.
 - ✓ Contiene las referencias a los “hijos”.
 - ✓ Implementa operaciones para manejar “hijos”.

✓ Consecuencias

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 así 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 deberían 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

Pattern Composite

- ✓ Hace mas fácil el agregado de nuevos tipos de componentes porque los clientes no tienen que cambiar cuando aparecen nuevas clases componentes.
- ✓ 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)