



## Descubriendo Patrones

- ✓ Como es el proceso de descubrimiento?
- ✓ Que tipo de observacion/abstraccion realizamos?

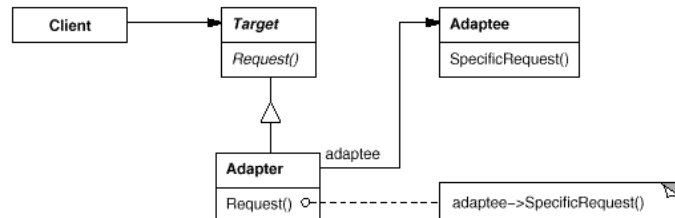


## Usando Patrones

- ✓ Supongamos que conocemos patrones(e.g. Adapter).
- ✓ Como mapeamos un patron a un diseño especifico?
- ✓ Como aplicamos el principio de Alexander ("use the patterns millions of times without doing the same thing twice")?



## Adapter

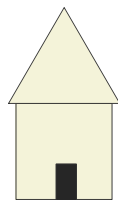


Como usamos esta informacion? Es suficiente?  
Que mas necesitamos?

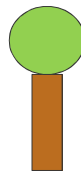


## Problema

- ✓ Supongamos que queremos manejar figuras compuestas y tratarlas como figuras simples (moverlas, rotarlas, etc).



Casa= Cuerpo + Techo  
Cuerpo= Puerta + Pared



Arbol= Tronco + Copa

Propiedad= Casa + Arbol



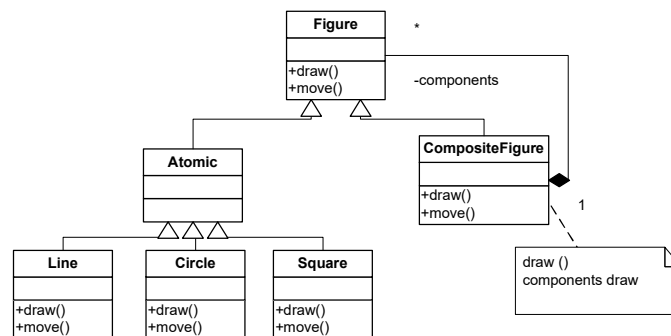
## Soluciones?

- ✓ Podemos tener un arreglo de figuras y marcarlas como partes de otras....
- ✓ Como seria tal arreglo cuando hay composiciones muy “profundas”
- ✓ Por ejemplo “Barrio”



## Una solucion 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

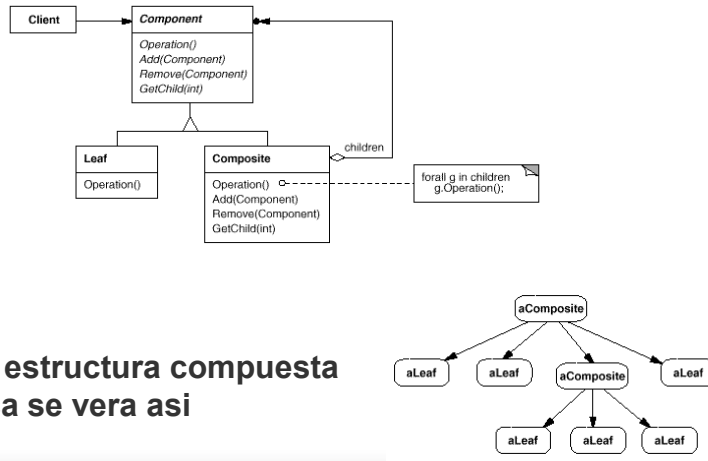
Use el patron Composite cuando

- ✓ 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 típica se vera asi



## Composite

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



## Pattern Composite

### ✓ 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



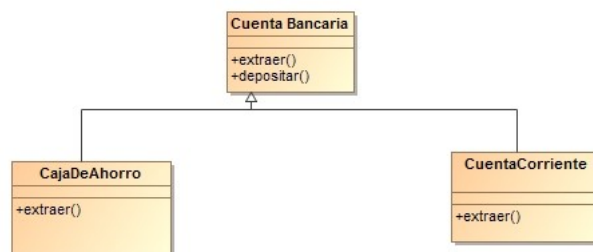
## Pattern Composite

- ✓ 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)



## Problema

- ✓ Supongamos una jerarquia de cuentas bancarias y una operación con “variantes” de acuerdo a la cuenta



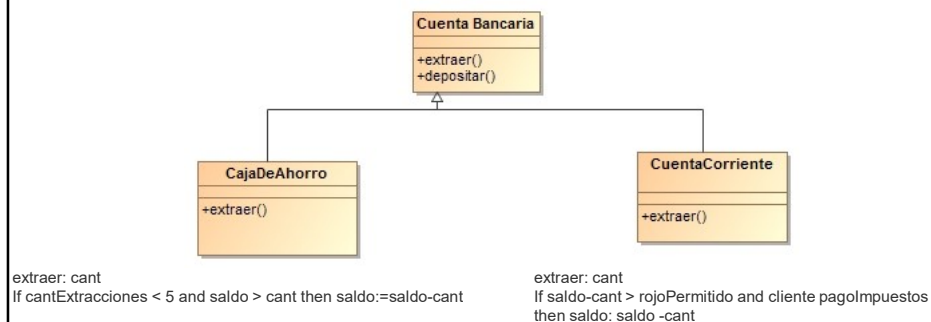


## El Metodo Extraer

- ✓ En la Clase Caja de Ahorro tiene que controlar el saldo contra 0 y la cantidad de extracciones
- ✓ En la Clase Cuenta Corriente tiene que controlar el saldo contra un “rojo” permitido y la situacion impositiva del cliente



## Solucion



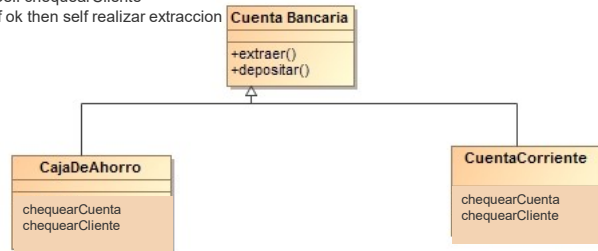
Problemas con esta Solucion





## Template Method

Extraer: cant  
Self chequearCuenta  
Self chequearCliente  
If ok then self realizar extraccion



## Template Method

### ✓ Intent:

- ✓ Definir el esqueleto de un algoritmo en un metodo, difiriendo algunos pasos a las subclases. El template method permite que las subclases re definas ciertos aspectos de un algoritmo sin cambiar su estructura

### ✓ Aplicabilidad

- ✓ Para implementar las partes invariantes de un algoritmo una vez y dejas que las sub-clases implementen los aspectos que varian



## ✓ Structure

