Orientacion a Objetos 2

Dra Alejandra Garrido

Dr. Alejandro Fernandez

Dr. Federico Balaguer

Dr. Gustavo Rossi

[garrido, alejandro.fernandez,federico.balaguer, gustavo]@lifia.info.unlp.edu.ar

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

Patterns-Definiciones

- Según Alexander

> *"A pattern describes a problem that occurs once and again in a context, and describe the core of the solution in such a way that it can be used millions of times without doing the same twice"*

Patrones de Diseño

Definicion (GOF):

A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue. It describes when it applies, whether it can be applied in view of other design constraints, and the consequences and trade-offs of its use.

Gamma, Helms, Johnson, Vlissides: Design Patterns. Elements of Reusable Object-Oriented Software.

Descripcion de un patron

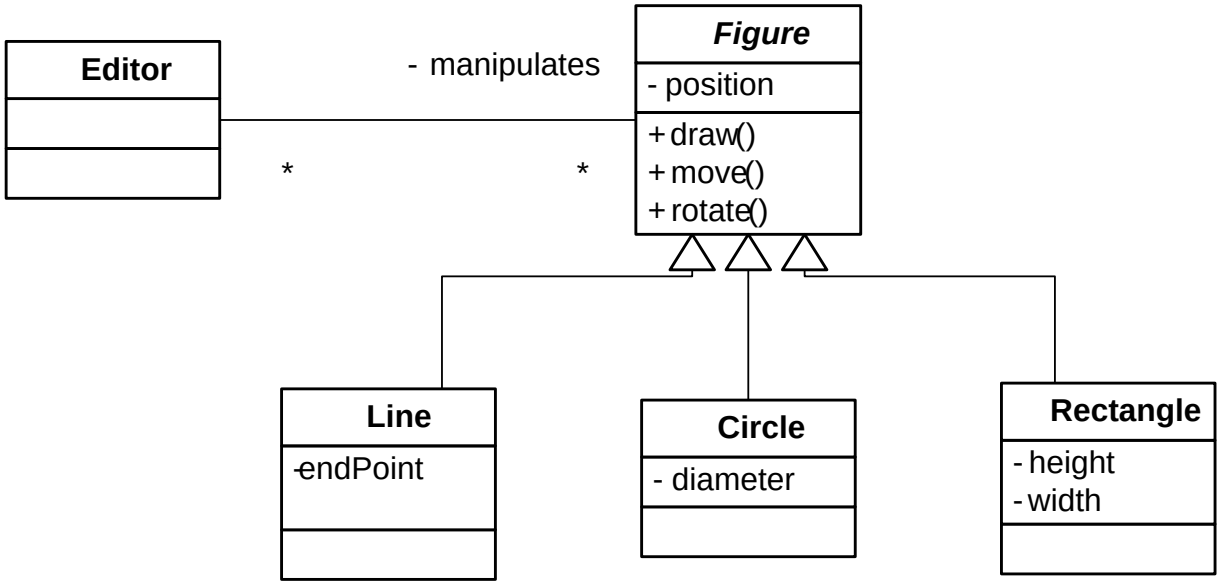
- Según GoF:
  - *Name*
  - *Intent*
  - *Motivation (Problem)*
  - *Aplicability*
  - *Structure*
  - *Participants*
  - *Collaborations*

Descripcion….

- ✓ *Consequences*
- ✓ *Implementatión*
- ✓ *Code*
- ✓ *Known Uses*
- ✓ *Related Patterns*

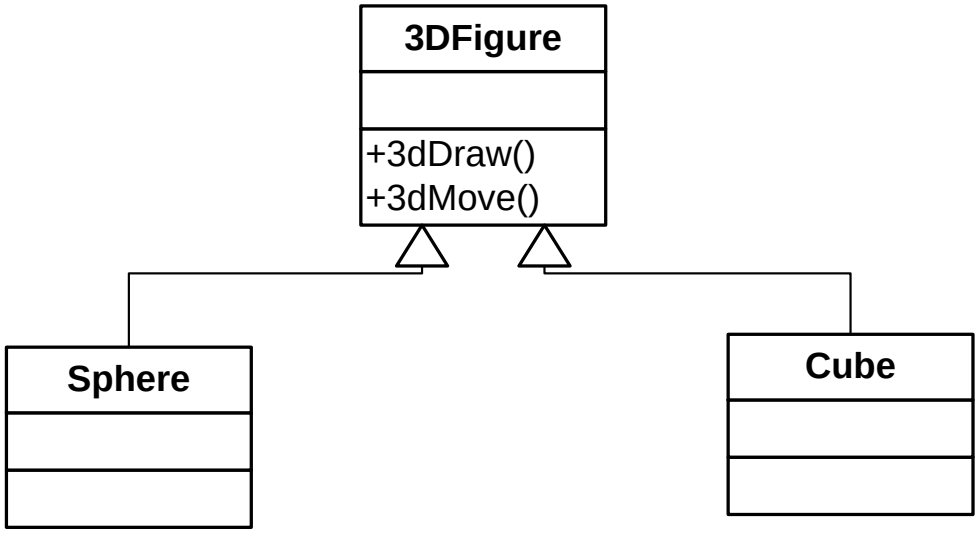
Ejemplo

- ✓ Editor grafico para manejar figuras geometricas:
- ✓ Observen polimorfismo en la relacion Editor-Figure



Ejemplo..

- ✓ Queremos extender el editor a figuras 3D



Problema

- ✓ Como integramos esta jerarquia?
- ✓ Que problemas tenemos?
- ✓ Sacrificamos polimorfismo?
- ✓ Editamos el codigo de la nueva jerarquia?

Soluciones....

- ✓ Cambiamos todos los nombres de los métodos de la jerarquía 3d a:

draw

Move

Etc...

Que pasa si conseguimos una nueva version?

Soluciones

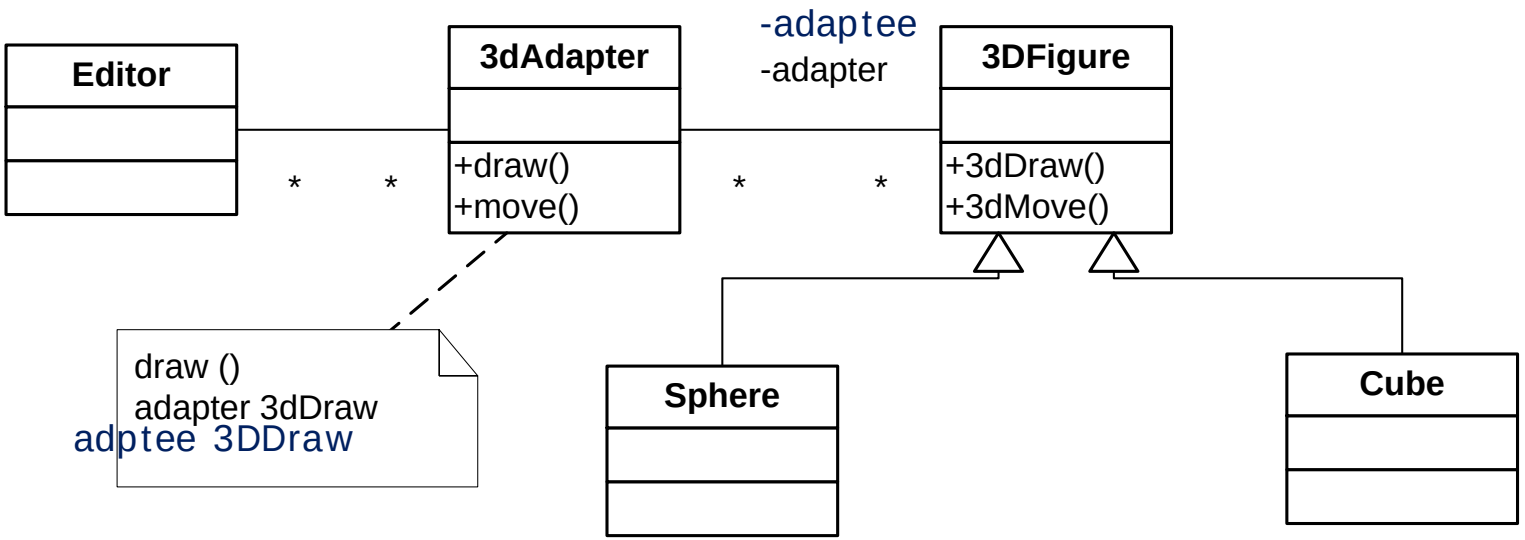
- ✓ En el código del editor metemos un if en todas las acciones que involucren un mensaje a figuras para decidir el mensaje que enviamos

- ✓ Que podría salir mal?

Solucion

Solucion

- ✓ Cuando tratamos con interfases incompatibles, intentar adaptarlas.



3dAdapter is sub-clase de?

Adapter

Intencion:

"Convertir" la interfaz de una clase en otra que el cliente espera EL Adapter permite que ciertas clases trabajen en conjunto cuando no podrian por tener interfaces incompatibles

Applicabilidad:

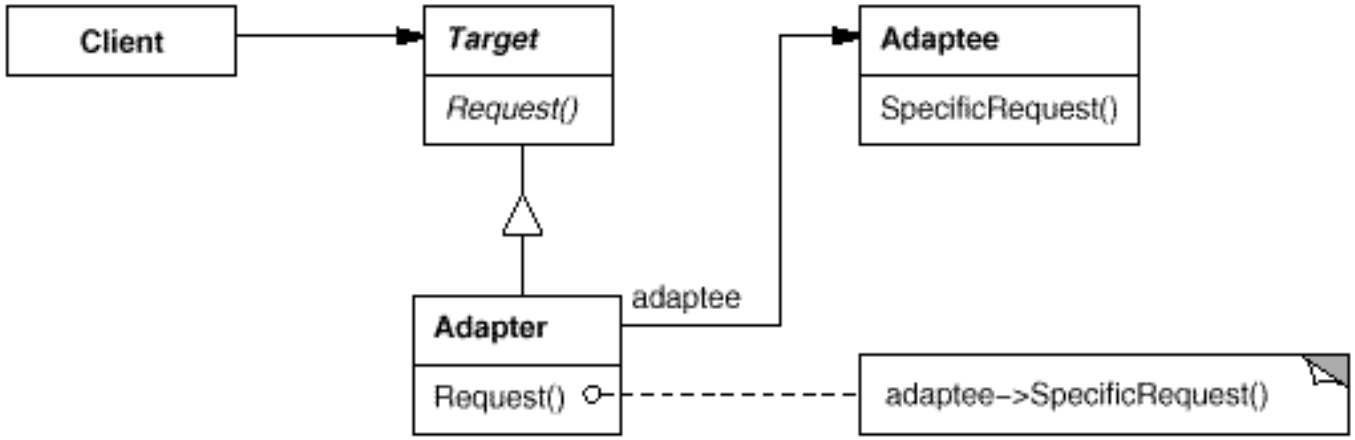
Use el adapter cuando:

- Ud quiere usar una clase existente y su interfaz no es compatible con lo que precisa

Adapter

✓ Estructura

target representa el mismo rol que representan las figuras
expresa el protocolo que el cliente utiliza que no es el portocolo qeu ustiliza el adaptado
adaptador: subclase del adaptador que va a conocer al adaptado para redireccionarle el mensaje adaptado



Como se lee/interpreta este diagrama?
Que representan las clases acá?

Adapter. Participants

- **Participantes:**
  - **Target** (Figure)
    - define the domain-specific interface that Client uses.
  - **Client** (Editor)
    - collaborates with objects conforming to the Target interface.
  - **Adaptee** (3DFigure)
    - defines an existing interface that needs adapting.
  - **Adapter** (3DAdapter)
    - adapts the interface of Adaptee to the Target interface.

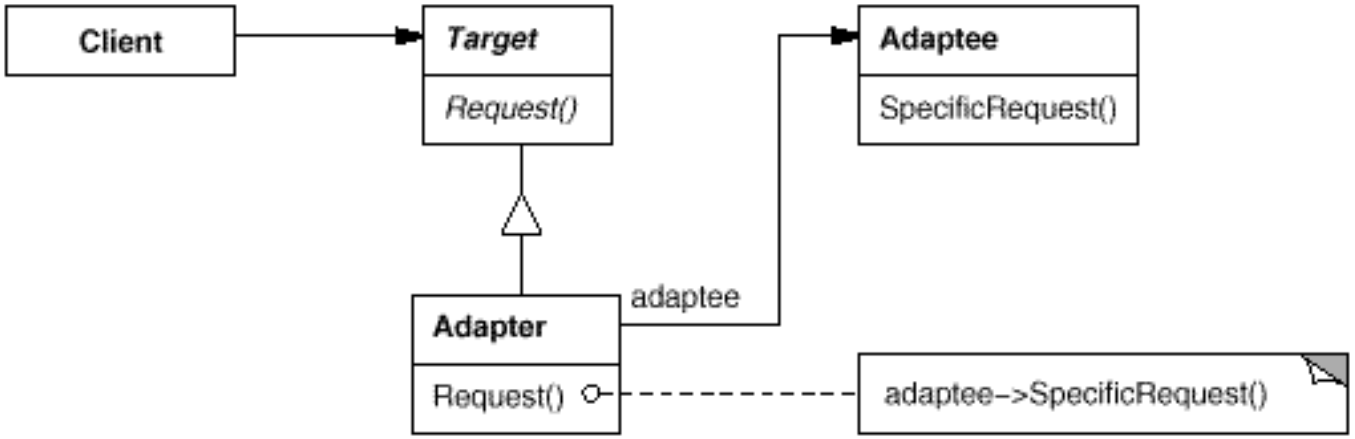
Descubriendo Patrones

- ✓ 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”)?

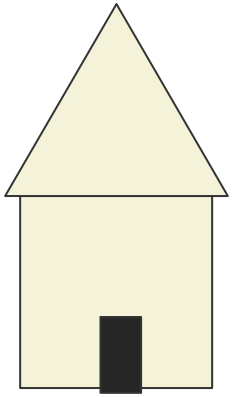
Adapter

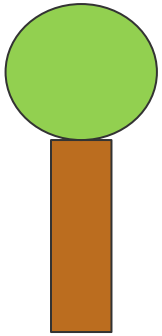


Como usamos esta informacion? Es suficiente?
Que mas necesitamos?

Problema

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





Casa= Cuerpo + Techo
Cuerpo= Puerta + Pared

Arbol= Tronco + Copa

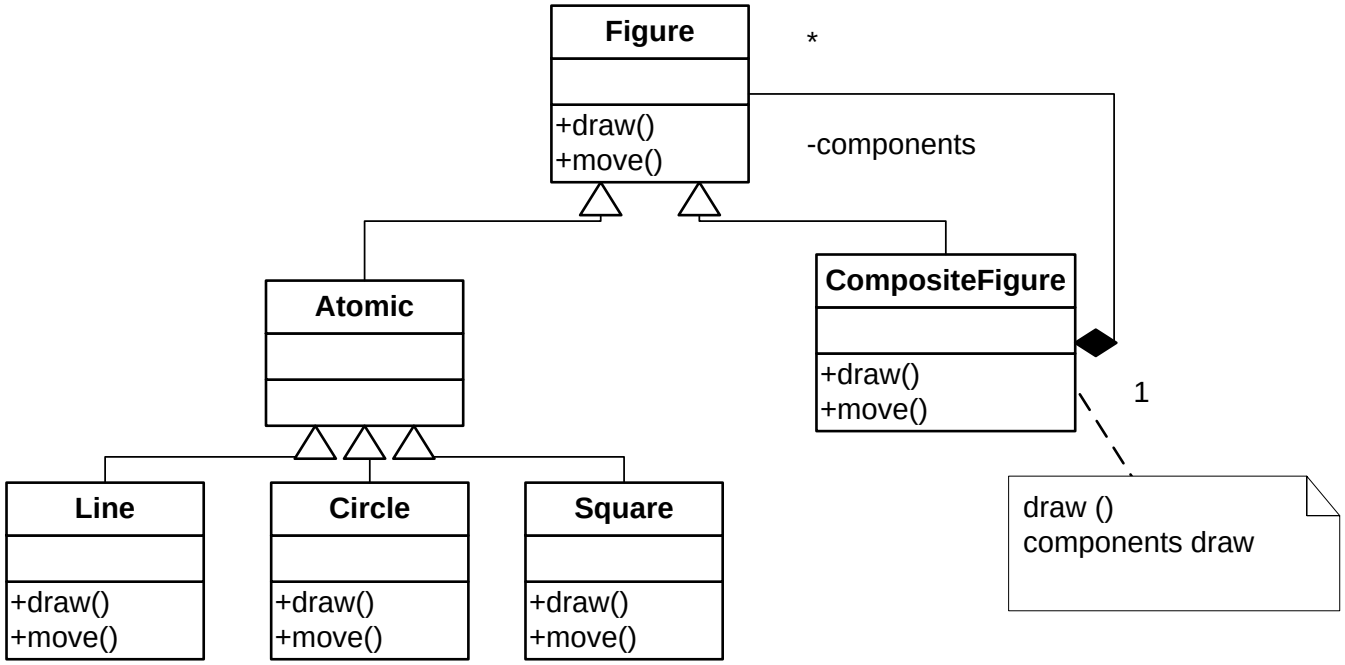
Propiedad= Casa + Arbol

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

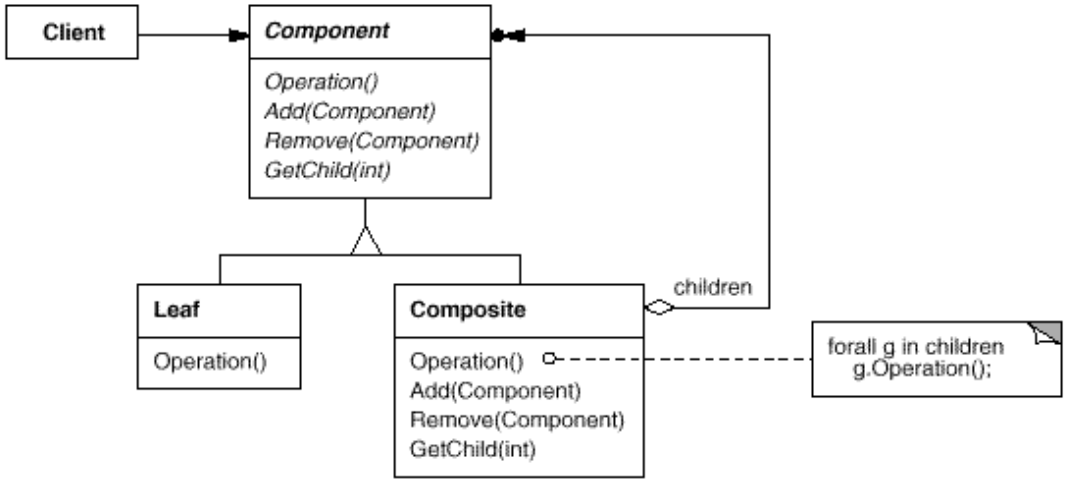
✓ Applicability

Use el patron Composite cuando

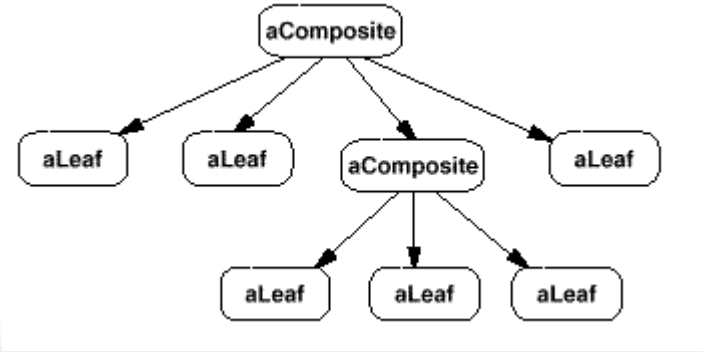
- ✓ quiere representar jerarquias parte-todo de objetos.
- ✓ quiere que los objetos “clientes” puedan ignorar las diferencias entre composiciones y objetos individuales. Los clientes trataran a los objetos atomicos y compuestos uniformemente.

Pattern Composite

✓Structure



Una estructura compuesta tipica 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

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

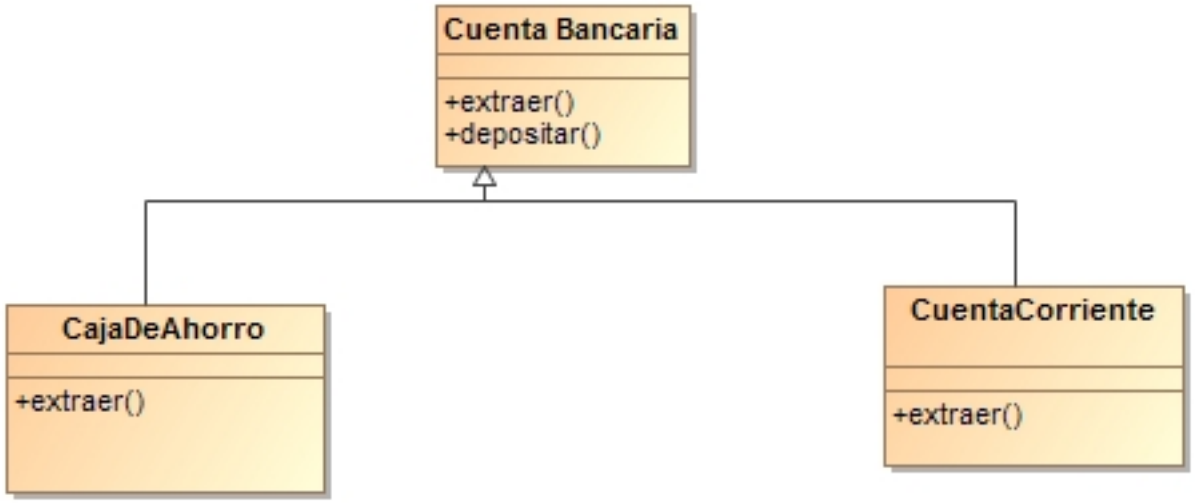
Pattern Composite

- ✓ 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

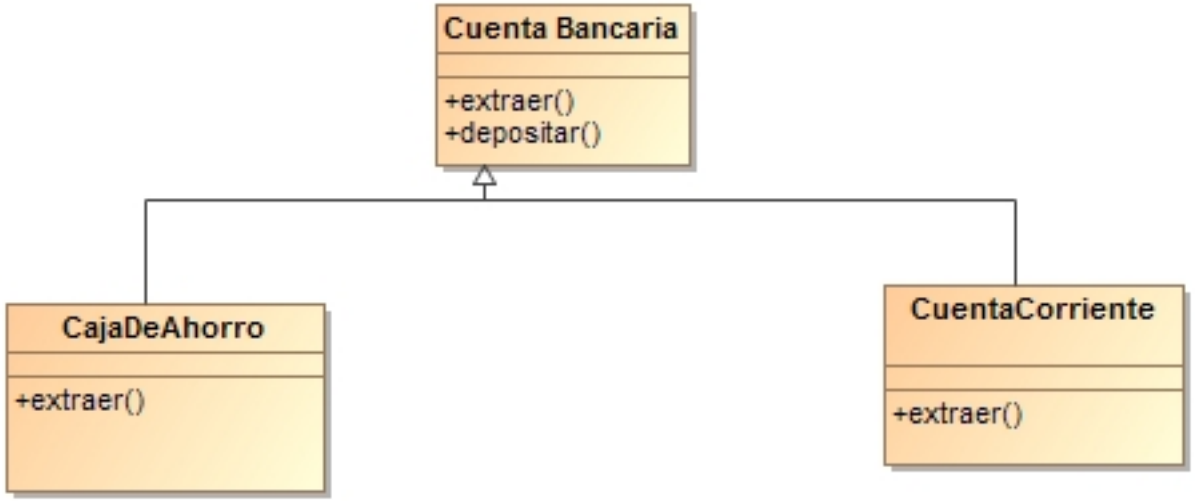


El Metodo Extraer

- ✓ 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



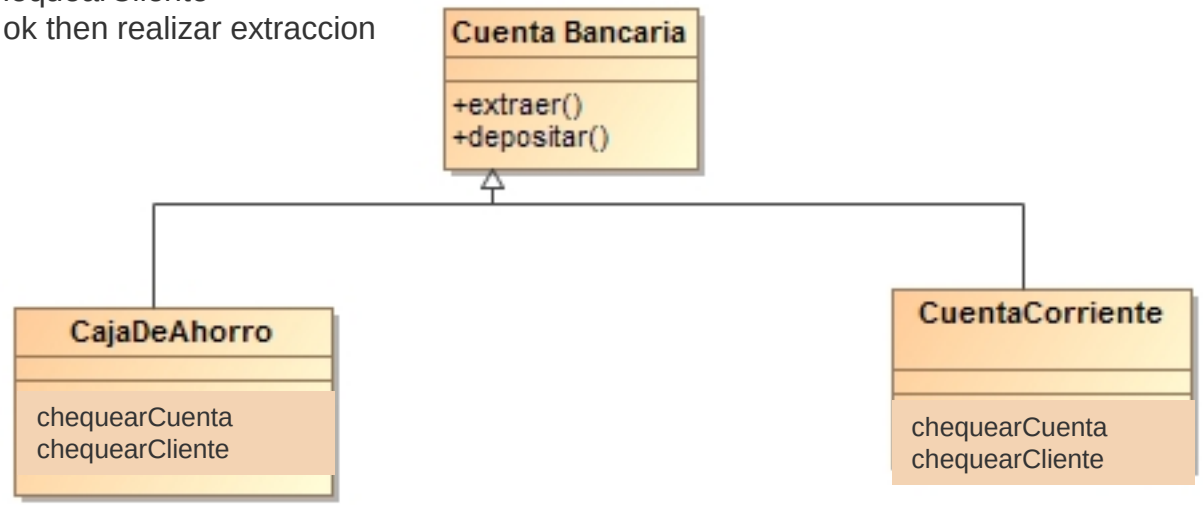
extraer (cant)
If cantExtracciones < 5 and saldo > cant then saldo=saldo-cant

extraer (cant)
If saldo-cant > rojoPermitido and cliente pagoImpuestos
then saldo=saldo -cant

Problemas con esta Solucion

Template Method

Extraer: cant
chequearCuenta
chequearCliente
If ok then 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

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

Template Method

- ✓ Structure