# PATRÓN DE DISEÑO **COMPOSITE**

SUNG PEI JUNG

Análisis y Diseño de Sistemas 2017

# ¿QUÉ ES?

- Es un patrón de diseño estructural.
- El patrón Composite nos permite construir objetos complejos partiendo de otros más sencillos utilizando una estrategia de composición recursiva.

# INTENCIÓN

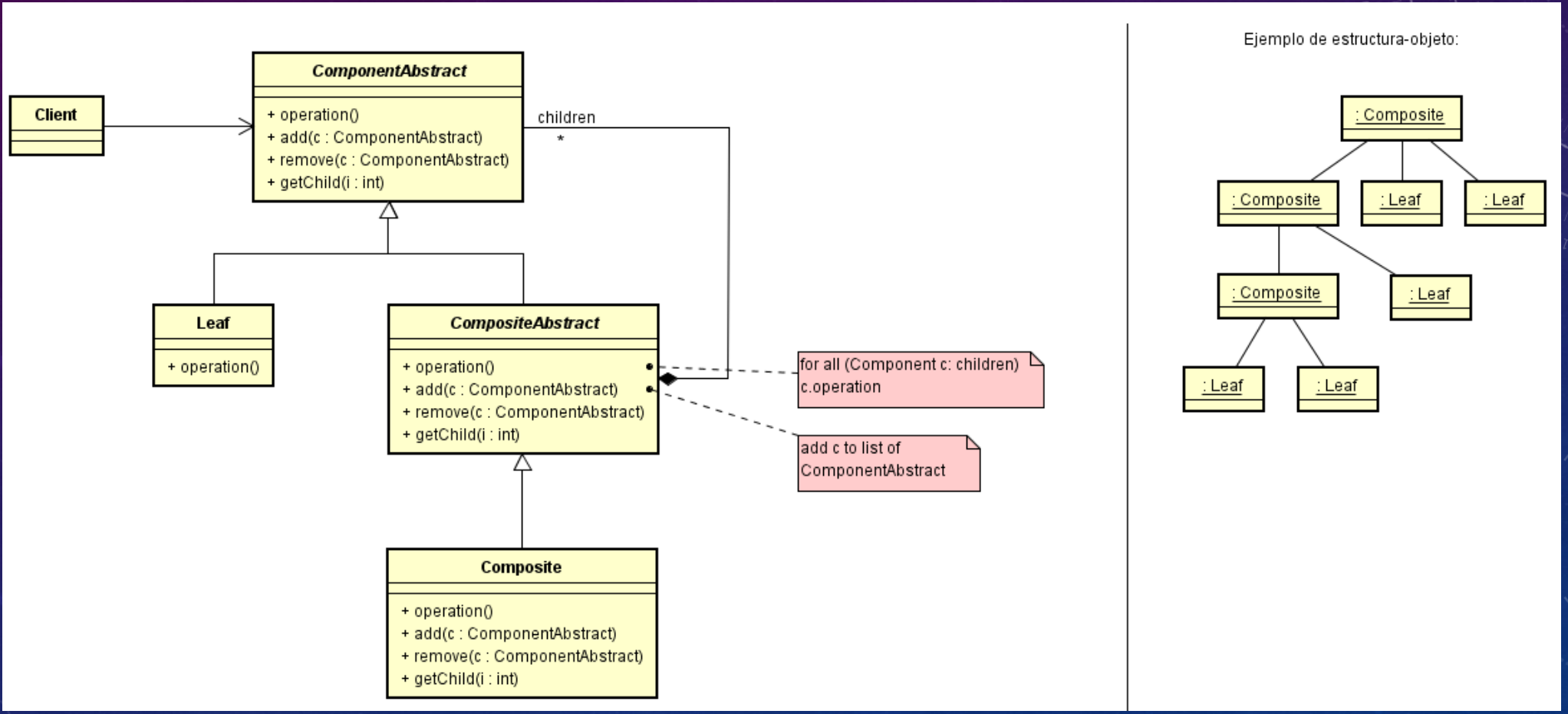
- Componer objetos en estructuras de árbol para representar jerarquías parte-todo.

- Permitir a los clientes tratar objetos individuales y objetos compuestos de una manera uniforme.

# MOTIVACION

- En aplicaciones gráficas los usuarios pueden construir figura complejos a partir de componentes simples (líneas, textos, triángulos, etc..) agrupándolos para formar otros mas grandes, que a su vez pueden estar agrupados para formar componentes todavía mas grandes.
- Una implementación simple de estas aplicaciones podría definir clases para gráficas primitivas y clases que actúan como contenedores de estas gráficas.
- El problema de este enfoque es que el código que usan estas clases deben tratar los objetos primitivos y contenedores de manera diferente.
- Como solución a este problema el patrón composite propone una clase abstracta que represente tanto objetos primitivos como contenedores y sus operaciones.

# ESTRUCTURA GENERAL

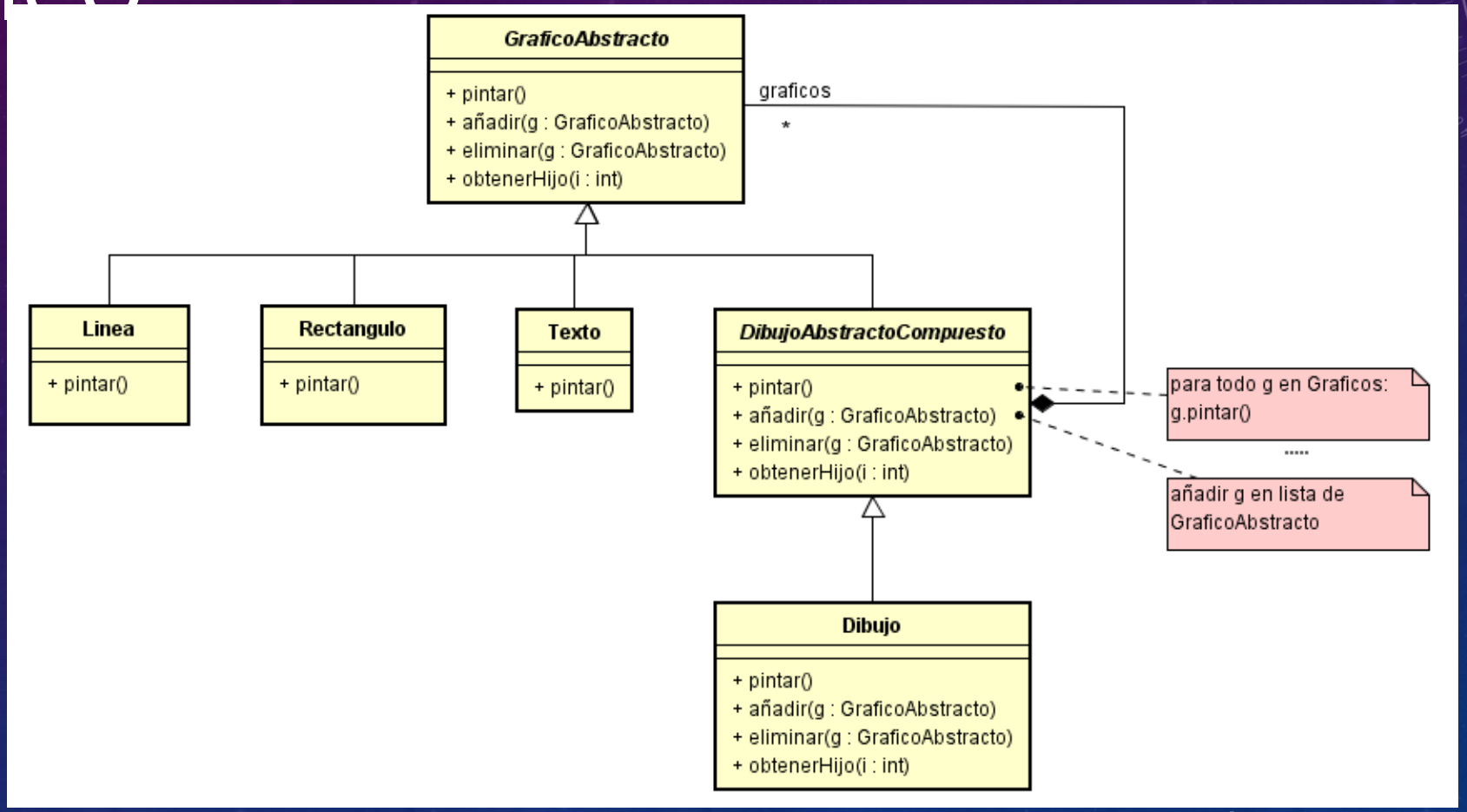



Análisis y Diseño de Sistemas 2017

# PARTICIPANTES

- Client:
  - ◊ Manipula objetos en la composición a través de la interface Component.
- Component (Ej: Grafico) :
  - ◊ Declara la interface para objetos en la composición
  - ◊ Implementa el comportamiento de la interface común a todas las clases.
  - ◊ Declara una interface para el acceso y manejo de sus componentes hijos.
- Leaf (Ej: Linea, Rectangulo, Texto) :
  - ◊ Representa objetos hoja en la composición. No tiene hijos.
  - ◊ Define el comportamiento de los objetos primitivos.
- Composite (Ej: Dibujo) :
  - ◊ Define el comportamiento de los componentes que tienen hijos.
  - ◊ Almacena componentes hijos.
  - ◊ Implementa las operaciones relacionadas a los hijos de la interface Component.

# EJEMPLO: DIAGRAMA DE CLASES DE GRÁFICO



# APLICABILIDAD

- Cuando se desea representar jerarquías parte-todo de objetos.

- Cuando se desea que los clientes ignoren la diferencia entre objetos compuestos y objetos individuales.

# CONSECUENCIAS

- Define jerarquías de clases formadas por objetos primitivos y compuestos. Si el código cliente espera un objeto simple, puede recibir también un objeto compuesto.
- Simplificación de código. Le permite al cliente tratar indistintamente a los objetos primitivos y compuestos, por lo general el cliente no sabe si trata con una hoja o un elemento compuesto.
- Facilidad al añadir nuevos componentes.
- Hace al diseño demasiado general. La desventaja de esto es que se hace mas difícil restringir el tipo de componentes de un objeto compuesto.

# USOS CONOCIDOS

- En el paquete java.awt.swing (clase para realizar interfaces gráficas en java)
  - Component
    - Component
  - Composite
    - Container (abstracta)
    - Panel (concreta)
    - Frame (concreta)
    - Dialog (concreta)
  - Leaf
    - Label
    - TextField
    - Button