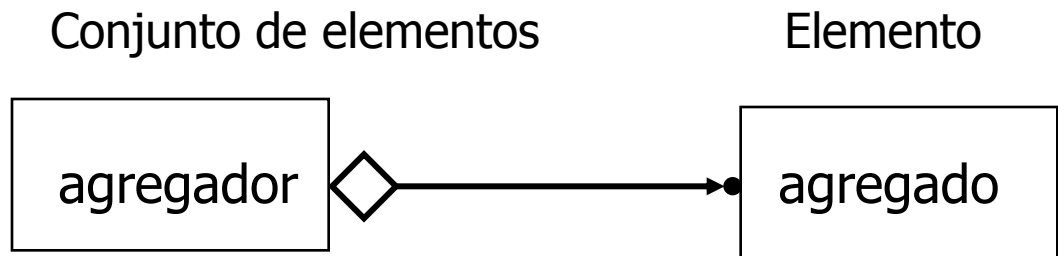
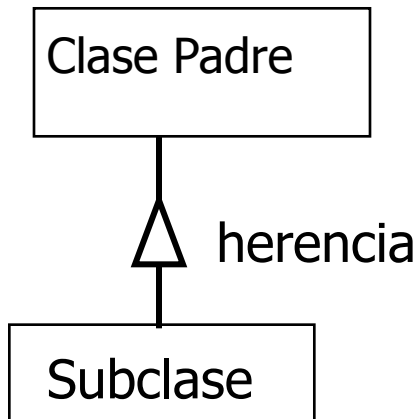
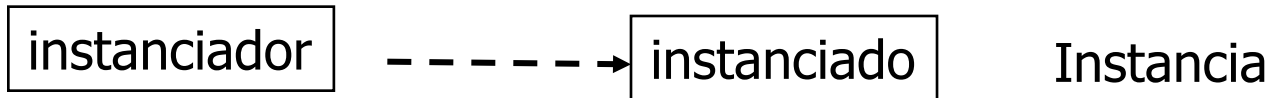
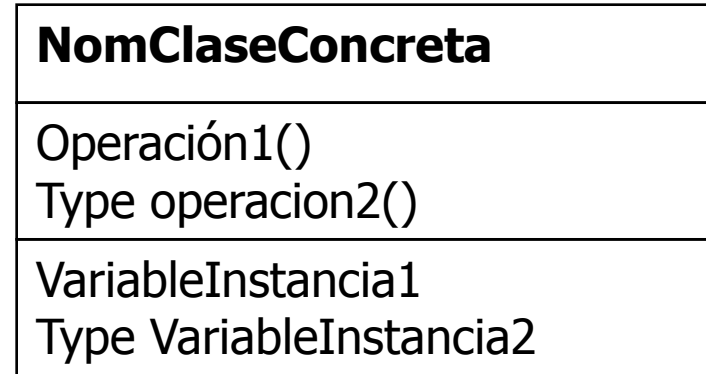
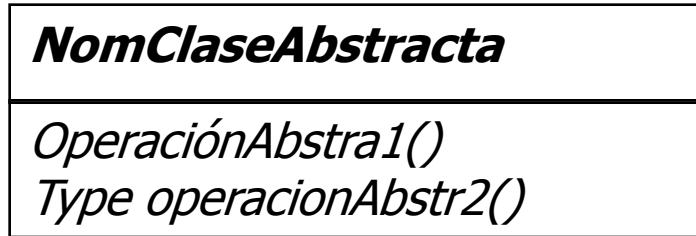
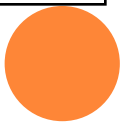


El Tema 4. El catálogo del patrones de diseño

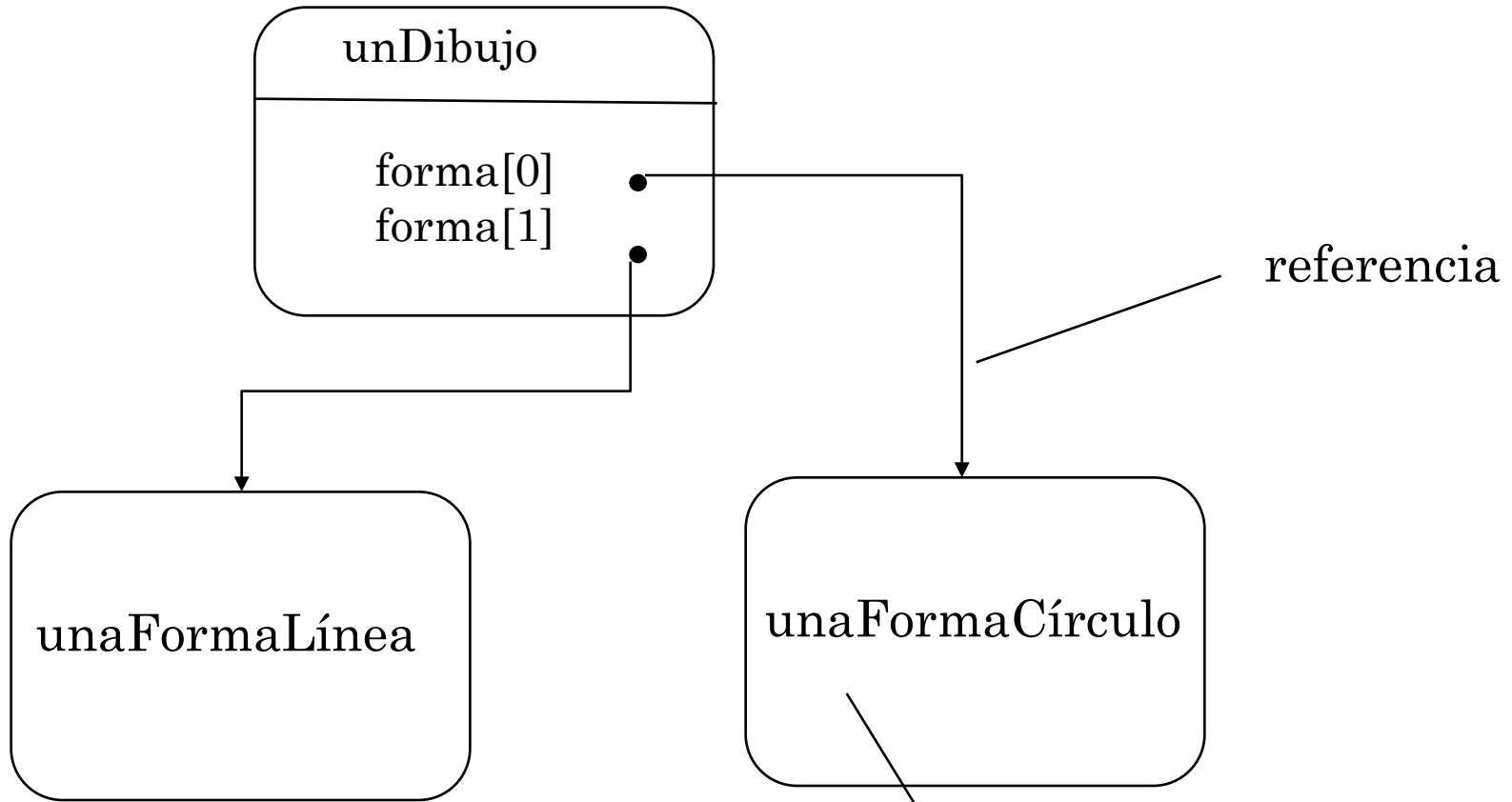
1. Las notaciones de OMT para clases



- Más que uno



- Las notaciones diagrama de objetos



Una instancia de la Clase `FormaCírculo`



3.3 El patrón Composite (Compuesto)

Propósito

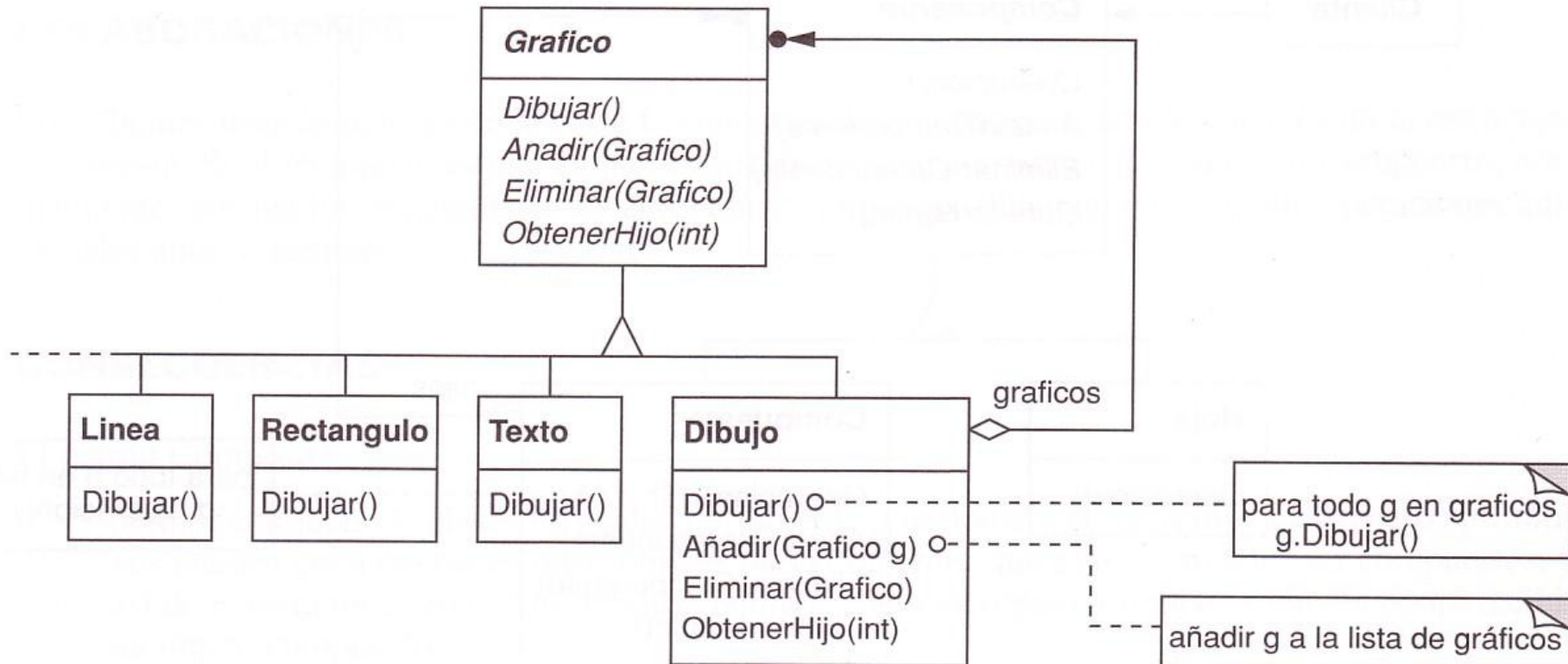
- Es un patrón para componer los objetos a una estructura de árbol para presentar una jerarquía de parte-todo.
- Los clientes pueden tratar los objetos individuales y los compuestos de ellos uniformemente.

Motivación

- El problema que trata
 - Como en un editor, los usuarios quieren constituir diagramas complejos desde componentes simples
 - Los objetos primitivos y los contenedores de ellos están en diferentes maneras, aunque los clientes tratan ambos en la misma manera
- La clave del patrón Composite:
Una clase abstracta que representa ambos los primitivos y sus contenedores .



Un ejemplo: en un sistema de gráficos. Primitivos y compuestos



La clase abstracta: Grafico: lleva la formación recursiva en su definición

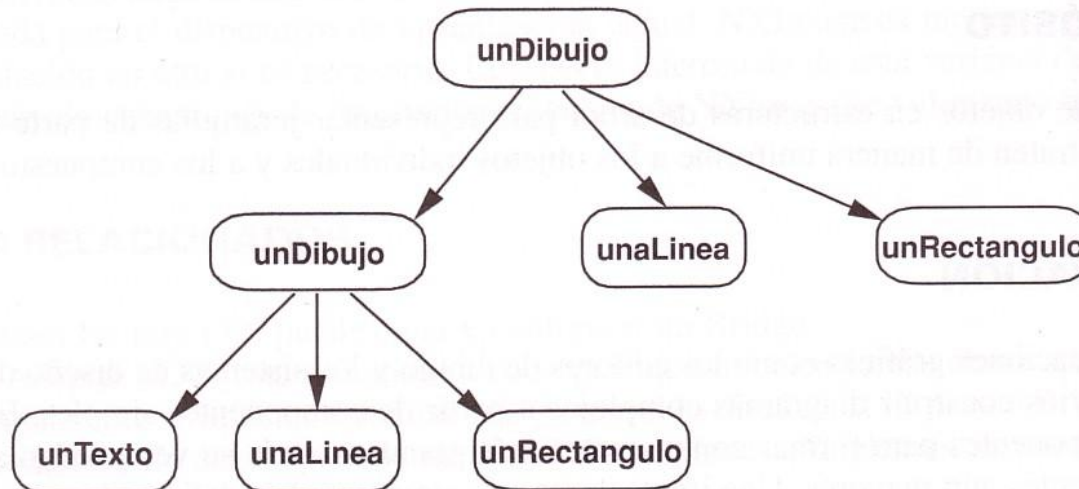
Sus operaciones son generales para sus subclasses

Sus subclasses :

- Gráficos primitivos: Linea, Rectangulo, Texto. Todas implementan la función Dibujar() para dibujarse.
- No implementan las funciones para administrar hijos

La subclase compuesta - Dibujo : define una agregación de objetos de Grafico.

- Implementa Dibujar() para que llame al Dibujar de sus hijos
- Implementa operaciones para administrar a sus hijos
- Una estructura de componer objetos compuestos recursivamente: puede tener hijos primitivos y compuestos como su padre Grafico

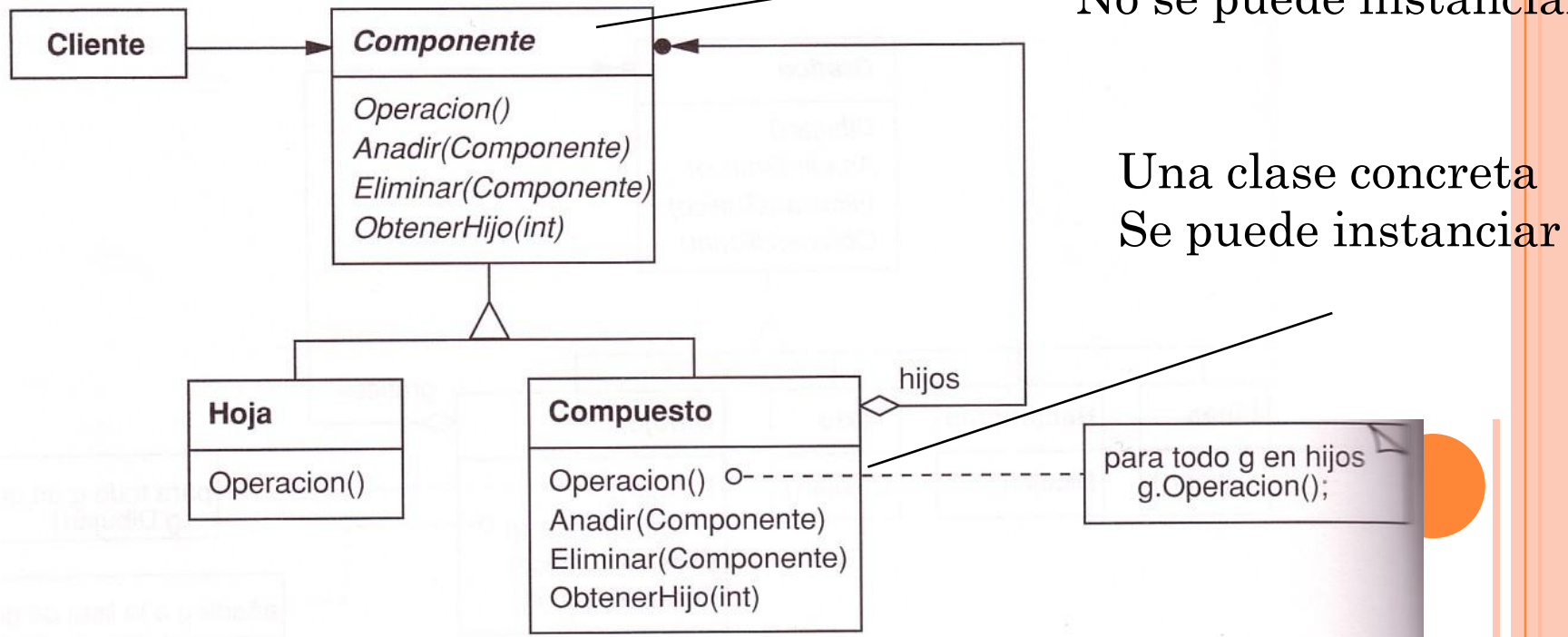


Aplicabilidad

- Cuando necesitamos representar la relación parte-todo
- Cuando queremos que los clientes olviden la diferencia entre las composiciones de objetos y los objetos individuales

Estructura

- Estructuras de las clases de Composite



Participantes (las clases que participan en el patrón Composite y sus responsabilidades)

1. Componente (como el Grafico en la parte de motivación)

- Declara la interfaz para los objetos en la composición que los clientes puedan acceder
- Implementa comportamientos por defectos para todas las clases
- Declara una interfaz para acceder y administrar sus componentes hijos

ObtenerHijo(int)

2. Hoja (Retangulo, Linea, Texto, etc)

- Representa los objetos hoja en la composición. No tienen hijos.
- Define el comportamiento de los objetos primitivos en la composición

Operacion()



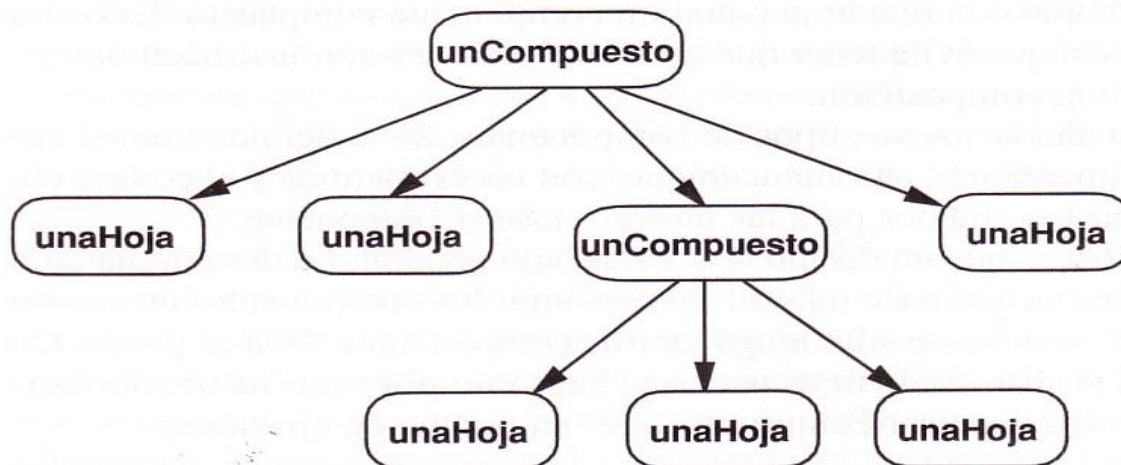
3. Compuesto (Dibujo)

- Define el comportamiento para los componentes que tienen hijos
- Almacena los componentes hijos
- Implementa operaciones relacionadas a los hijos en la interfaz de Componente

4. Cliente

- Manipula los objetos en la composición a través de la interfaz de la clase Componente

Estructuras típica de los objetos de Composite



Colaboraciones

- Existen colaboraciones entre el cliente y Componente
 - El cliente utiliza la interfaz para interactuar con los objetos en la composición
 - Si el receptor es una hoja: la petición se trata directamente
 - Si el receptor es un Compuesto: la petición se pasa a sus hijos

Puede tener más operaciones extras

Consecuencias

- Se define una jerarquía de clases con objetos primitivos y compuestos
- Simplifica las tareas del cliente. Puede tratar un compuesto como un objeto primitivo
- Facilita añadir nuevos componentes sin modificar la estructura ni los códigos de clientes
- Desventajas:
 - El diseño de programa puede ser demasiado general
 - Difícil de restringir los componentes de un compuesto. - El compilador no hace el trabajo con verificación de tipo. Eres tú que tienes que verificarlos en la hora de ejecución.



Implementación

Temas para considerar en la implementación:

1) Referencias explícitas al padre

2) Compartir componentes

- Para ahorrar espacio de almacenamiento
- Será difícil a compartir si el componente tiene un solo padre
- La solución aparece en el patrón Flyweight

3) Maximizar la interfaz de Componente

- Es bueno a definir más operaciones posibles en la clase Componente tanto para Compuesto como para Hoja
- Define operaciones por defectos en Componente y luego las subclases puedan redefinirlas si son necesarias.



4) Dónde declarar las operaciones para administrar los hijos

Es una decisión de equilibrio entre la seguridad y transparencias

- Definir en la raíz (Componente): mejor transparencia peor seguridad porque los clientes pueden hacer cosas raras
- Definirlas en Compuesto: mejor seguridad pero pierde la transparencia
- Una solución: verificar en la interfaz si lo que pide es un Compuesto o una hoja.

```
class Compuesto;  
class Componente {  
public: // ...  
    virtual Compuesto* ObtenerCompuesto() { return 0; }  
}
```

```
class Compuesto : public Componente {  
public:  
    void Anadir(Componente*);  
    //...
```



```
virtual Compuesto* ObtenerCompuesto ()  
    { return this;}  
};
```

```
class Hoja : public Componente {  
//...  
};
```

Códigos de muestras

- Ejemplos: la organización de equipos como un ordenador, componentes de estéreos etc. Se puede modelar con Compuesto
Un chasis: puede contener drives de discos duros, placas planas
Un bus: puede contener tarjetas
Un armario: puede contener chasis, buses etc.
- La jerarquía parte-todo de la clase Equipo con su interfaz



```
class Equipo {
public:
    virtual ~Equipo(); // el destructor
    // miembros inspectores
    const char* Nombre() { return _nombre;}

    virtual Vatio Potencia();
    virtual Moneda PrecioNeto();
    virtual Moneda PrecioConDescuento();
    // facilidades
    virtual void Anadir(Equipo*);
    virtual void Eliminar(Equipo*);
    virtual Iterador<Equipo*>* CrearIterador();
    // crea iterador para acceder las subpartes del equipo
protected:
    Equipo(const char*); // el constructor
Private:
    const char* _nombre;
};
```



Las subclases del Equipo:

- Hojas: los equipos tales el drive de disco duro, un circuito integrado, un interruptor
- Un compuesto: EquipoCompuesto que es la base para todos los que contienen otros equipos como sus sub-partes.

Una hoja: Disquetera

```
class Disquetera : public Equipo {
public:
    Disquetera(const char*);
    virtual ~Disquetera(); // el destructor
    // miembros inspectores
    const char* Nombre() { return _nombre;}

    virtual Vatio Potencia();
    virtual Moneda PrecioNeto();
    virtual Moneda PrecioConDescuento();
    // facilidades

};
```

