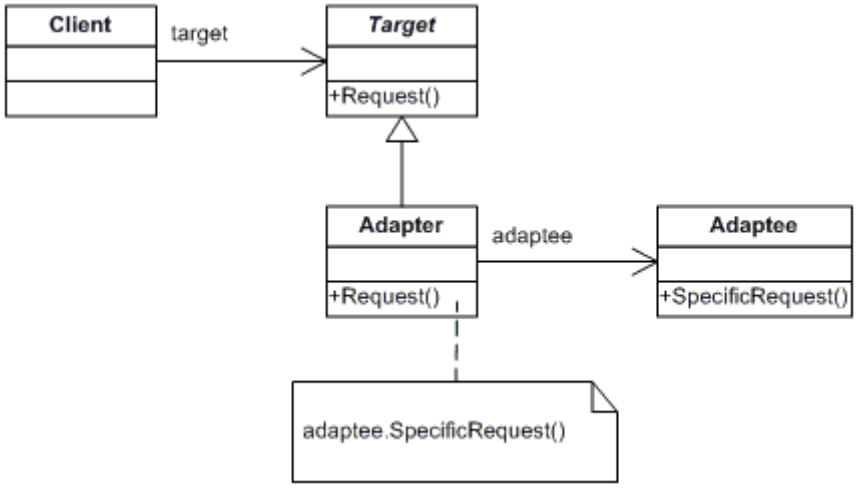
**PATRONES DE DISEÑO ESTRUCTURALES**

- **Adapter** (Frecuencia de uso: Media alta)

Convierte la interfaz de una clase en otra interfaz que los clientes esperan. El adaptador permite que las clases trabajen juntas y que no podrían hacerlo de otra manera debido a interfaces incompatibles.

Se mapea la interfaz de una clase sobre otra, para que puedan trabajar juntas.




---

Código estructural

Salida

Called SpecificRequest()

- Ejemplo

Uso de un banco de datos químico heredado. Objetos compuestos químicos que acceden al banco a través de una interfaz Adapter.

Participantes

Las clases y objetos que participan en este patrón son:

- Target (ChemicalCompound)
  - define la interfaz específica de dominio que utiliza el cliente.
- Adapter (Compound)
  - adapta la interfaz Adaptee a la interfaz de destino.

  Variables descriptivas de componentes

- Adaptee (ChemicalDatabank)
  - define una interfaz existente que necesita adaptarse.

Datos de componentes.

- Client (AdapterApp)
    - colabora con objetos conformes a la interfaz Target.

Salida

Compound: Unknown ———

Compound: Water ———
Formula: H20
Weight : 18.015
Melting Pt: 0
Boiling Pt: 100

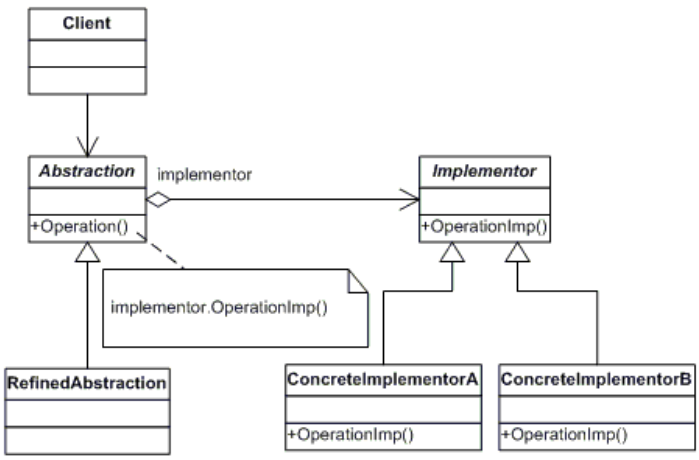
Compound: Benzene ———
Formula: C6H6
Weight : 78.1134
Melting Pt: 5.5
Boiling Pt: 80.1

Compound: Alcohol ———
Formula: C2H6O2
Weight : 46.0688
Melting Pt: -114.1
Boiling Pt: 78.3

- **Bridge** (Frecuencia de uso: Media)

Desacopla una abstracción de su implementación para que los dos puedan variar independientemente.

Separa/desacopla una interfaz de su implementación. Así, esta implementación puede evolucionar sin cambiar los clientes que utilizan la abstracción del objeto.



Código estructural

Salida

ConcreteImplementorA Operation

ConcreteImplementorB Operation

- Ejemplo

Abstracción de un obj de negocio que se va a desacoplar de la implementación en un obj DataObject. La implementaciones de DataObject van a poder evolucionar dinámicamente sin cambiar ningún cliente.

Participantes

Las clases y objetos que participan en este patrón son:

- Abstraction (BusinessObject)
  - Define la interfaz de la abstracción.
  - Mantiene una referencia a un objeto de tipo Implementor.
- RefinedAbstraction (CustomersBusinessObject)
  - Amplía la interfaz definida por
- Implementor (DataObject)
  - Define la interfaz para las clases de implementación. Esta interfaz no tiene que corresponder exactamente con la interfaz de Abstraction; de hecho, las dos interfaces pueden ser bastante diferentes. Normalmente, la interfaz de Implementación solo proporciona operaciones primitivas, y la Abstracción define operaciones de nivel superior basadas en estas primitivas.
- ConcreteImplementor (CustomersDataObject)
  - Implementa la interfaz del implementador y define su implementación concreta.

Salida

Jim Jones

Samual Jackson

Allen Good

Customer Group: Chicago

Jim Jones

Samual Jackson

Allen Good

Ann Stills

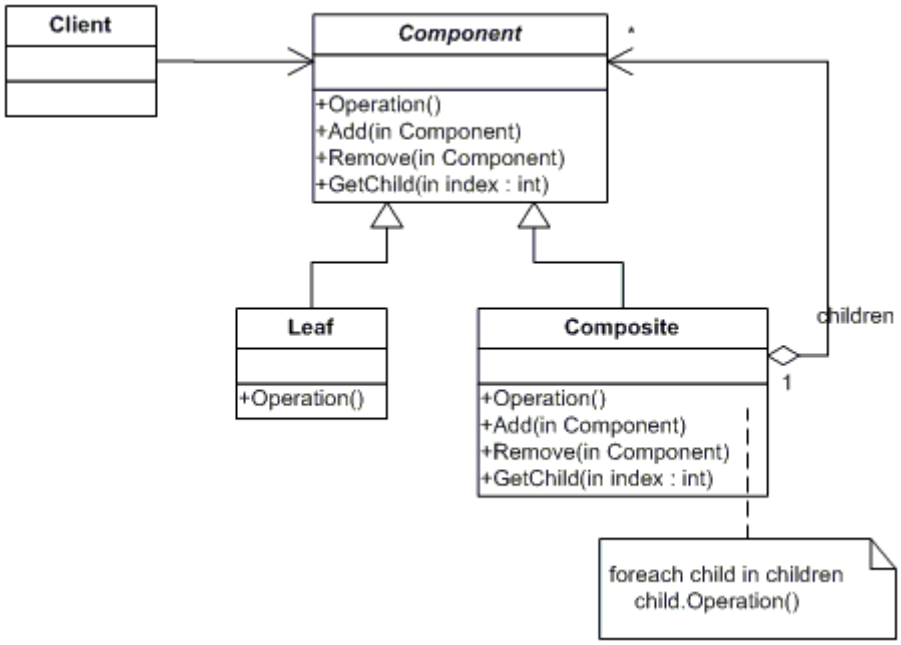
Lisa Giolani

Henry Velasquez

- **Composite** (Frecuencia de uso: Medio alta)

Compone los objetos en estructuras de árbol para representar jerarquías enteras. Permite a los clientes tratar objetos individuales y composiciones de objetos de manera uniforme.

Client a través de la interfaz Component crea estructura de árbol y añade nodos. Composite nodo padre, Leaf nodos hijo.




Código estructural

Salida

-root

Leaf A

Leaf B

Composite X

Leaf XA

Leaf XB

Leaf C

- Ejemplo

  Estructura de árbol formada por nodos primitivos (líneas, círculos, …) y nodos compuestos (grupos de elementos de dibujo que forman elementos más complejos). Client compone elementos.

Participantes

Las clases y objetos que participan en este patrón son:

- Component (DrawingElement)
  - declara la interfaz para los objetos en la composición.
  - implementa el comportamiento predeterminado para la interfaz común a todas las clases, según corresponda.
  - declara una interfaz para acceder y administrar sus componentes secundarios.
  - (opcional) define una interfaz para acceder al padre de un componente en la estructura recursiva, y la implementa si es apropiado.
- Leaf (PrimitiveElement)
  - representa objetos Leaf en la composición. Un objeto Leaf no tiene hijos.
  - define el comportamiento de los objetos primitivos en la composición.
- Composite (CompositeElement)
  - define el comportamiento de los componentes que tienen hijos.
  - almacena componentes secundarios.
  - implementa operaciones hijo relacionadas en la interfaz Component.
- Client (CompositeApp)
  - manipula objetos en la composición a través de la interfaz de Component.

Salida

-+ Picture

Red Line

Blue Circle

Green Box

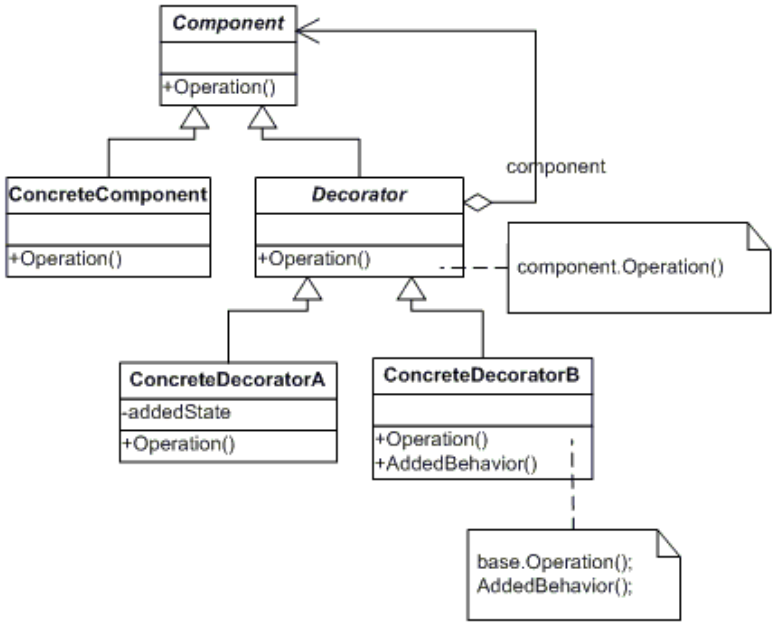
+ Two Circles

Black Circle

White Circle

- **Decorator** (Frecuencia de uso: Media)

Asigna responsabilidades adicionales a un objeto dinámica mente. Ofrecen una alternativa flexible a las subclases para extender la funcionalidad.




Código estructural

Salida

ConcreteComponent.Operation()

ConcreteDecoratorA.Operation()

ConcreteDecoratorB.Operation()

- Ejemplo

Agrega una funcionalidad de préstamo a los elementos que existen en una biblioteca.

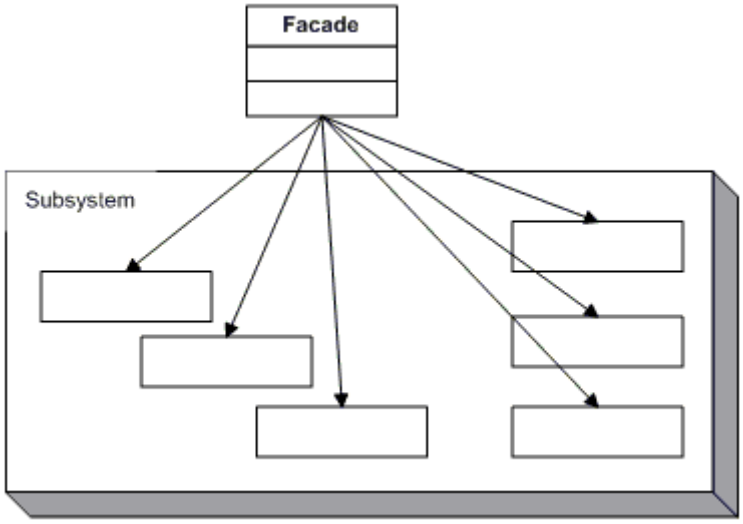
Las clases y objetos que participan en este patrón son:

- Component (LibraryItem)
  - define la interfaz para objetos que pueden tener responsabilidades agregadas dinámicamente.
- ConcreteComponent (Book, Video)
  - define un objeto al que se pueden añadir responsabilidades adicionales.
- Decorator (Decorator)
  - mantiene una referencia a un objeto Component y define una interfaz que se ajusta a la interfaz del Component.
- ConcreteDecorator (Borrowable)
  - agrega responsabilidades al Component.

- **Facade** (Frecuencia de uso: Alta)

Proporciona una interfaz unificada a un conjunto de interfaces en un subsistema. Define una interfaz de nivel superior que hace que el subsistema sea más fácil de usar.

Las clases del subsistema no implementan Facade.




Código estructural

Salida

MethodA() ——

SubSystemOne Method

SubSystemTwo Method

SubSystemFour Method

MethodB() ——

SubSystemTwo Method

SubSystemThree Method

- Ejemplo

Obtener solvencia crediticia de un solicitante. Mortgage (fachada) gestiona las clases del subsistema, delega en ellos (en sus métodos propios) para obtener el resultado a partir de una cantidad. El cliente crea Mortgage sólo, a ninguna clase del subsistema.

Participantes

Las clases y objetos que participan en este patrón son

- Facade (MortgageApplication)

    - sabe qué clases de subsistemas son responsables de una solicitud.
    - delega las solicitudes de los clientes a los objetos de subsistema apropiados.
- Subsystem classes (Bank, Credit, Loan)
    - Implementar la funcionalidad del subsistema.
    - manejar el trabajo asignado por el objeto Facade.
    - No tiene conocimiento del Facade y no hace ninguna referencia.

Output

Ann McKinsey applies for $125,000.00 loan

Check bank for Ann McKinsey

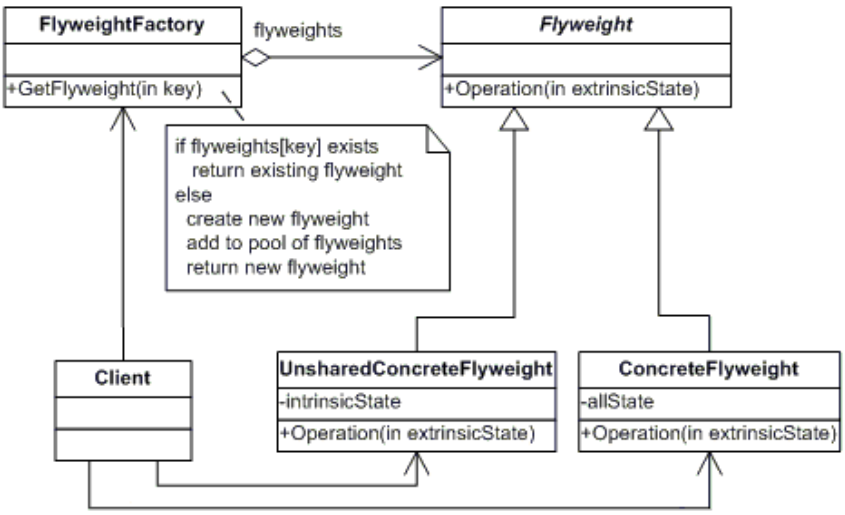
Check loans for Ann McKinsey

Check credit for Ann McKinsey

Ann McKinsey has been Approved

– **Flyweight** (Frecuencia de uso: Baja)

Soporta una gran cantidad de objetos pequeños de manera eficiente.




Código estructural

Salida

ConcreteFlyweight: 21

ConcreteFlyweight: 20

ConcreteFlyweight: 19

UnsharedConcreteFlyweight: 18

- Ejemplo

Un número pequeño de obj se va a compartir muchas veces con un documento que va a contener potencialmente muchos caracteres (procesamiento de textos).

Participantes

Las clases y objetos que participan en este patrón son:

- Flyweight (Character)
  - declara una interfaz a través de la cual los Flyweight pueden recibir y actuar en estado extrínseco.
- ConcreteFlyweight (CharacterA, CharacterB, …, CharacterZ)
  - implementa la interfaz de Flyweight y agrega almacenamiento para el estado intrínseco, si lo hay. Un objeto ConcreteFlyweight debe ser compartible. Cualquier estado que almacene debe ser intrínseco, es decir, debe ser independiente del contexto del objeto ConcreteFlyweight.
- UnsharedConcreteFlyweight (no usado en el ejemplo)
  - no todas las subclases Flyweight necesitan ser compartidas. La interfaz de Flyweight permite compartir, pero no es obligatorio. Es común que los objetos UnsharedConcreteFlyweight tengan objetos ConcreteFlyweight como hijos en algún nivel en la estructura de objetos Flyweight
- FlyweightFactory (CharacterFactory)
  - crea y maneja objetos Flyweight
  - se asegura que el Flyweight se comparte correctamente. Cuando un cliente solicita un Flyweight, el objeto FlyweightFactory crea una instancia existente o la crea, si no existe.
- Client (FlyweightApp)
  - mantiene una referencia al o los Flyweight(s).
  - calcula o almacena el estado extrínseco del Flyweight

Salida

A (pointsize 11)

A (pointsize 12)

Z (pointsize 13)

Z (pointsize 14)

B (pointsize 15)

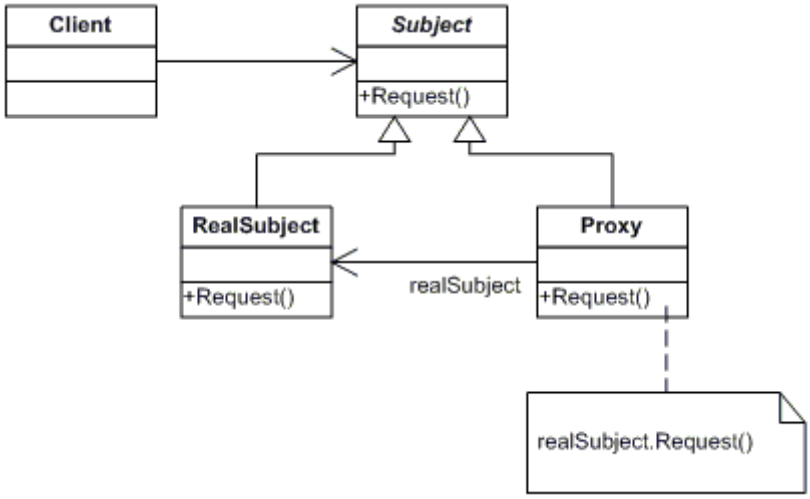
B (pointsize 16)

Z (pointsize 17)

B (pointsize 18)

– **Proxy** (Frecuencia de uso: Media alta)

Proporciona un sustituto o un marcador de posición para que otro objeto controle el acceso a él.




Código estructural

Salida

Called RealSubject.Request()

➢ Ejemplo

Objeto real Math codifica operaciones matemáticas definidas en interfaz IMath. Proxy crea obj Math con los que crea métodos de las funciones de operaciones. En Client se llama a Proxy para obtener resultados pasando los valores.

Participante

Las clases y objetos que participan en este patrón son:

- Proxy (MathProxy)
  - mantiene una referencia que le permite al proxy acceder al sujeto real. El proxy puede referirse a un Subject si las interfaces de RealSubject y Subject son las mismas.
  - proporciona una interfaz idéntica a la de los Subjects para que un proxy pueda ser sustituido por el Subject real.
  - controla el acceso al Subject real y puede ser responsable de crearlo y eliminarlo.
  - tras responsabilidades dependen del tipo de proxy:
    - _remote proxies_son responsables de codificar una solicitud y sus argumentos y de enviar la solicitud codificada al Subject real en un espacio de direcciones diferente.
    - _virtual proxies_pueden almacenar en caché información adicional sobre el Subject real para que puedan posponer su acceso.
    - _protection proxies_comprueban que la persona que llama tiene los permisos de acceso necesarios para realizar una solicitud.

- Subject (IMath)
  - define la interfaz común para RealSubject y Proxy para que se pueda usar un Proxy en cualquier lugar donde se espera un RealSubject.
- RealSubject (Math)
  - define el objeto real que representa el proxy.

Salida

4 + 2 = 6

4 - 2 = 2

4 * 2 = 8

4 / 2 = 2