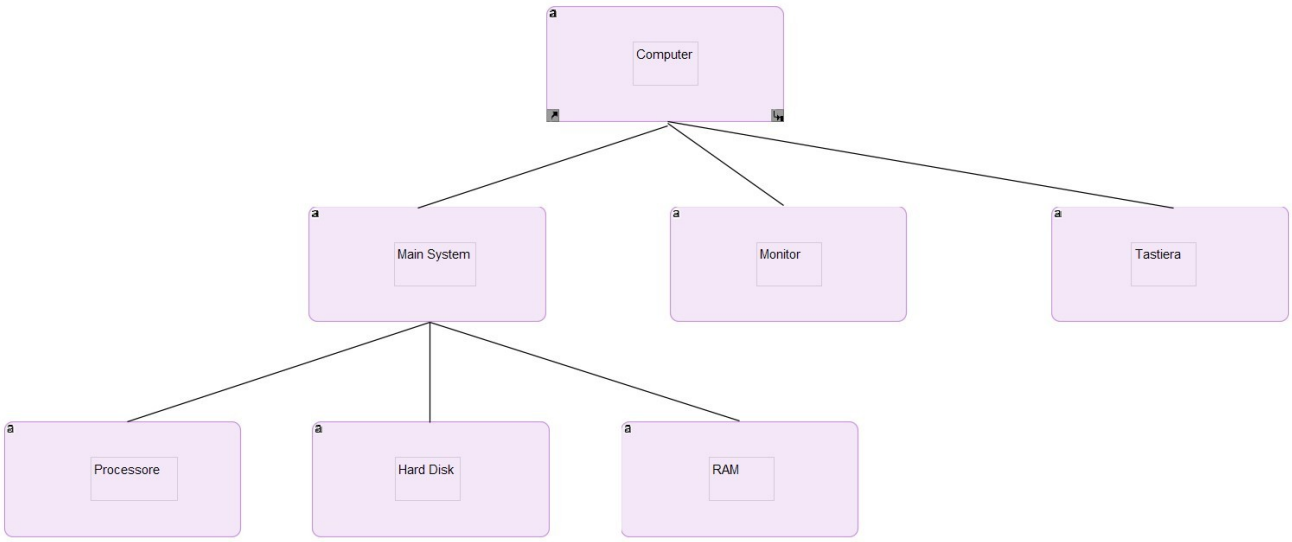
Composite

Consente la costruzione di gerarchie di oggetti composti. Gli oggetti composti possono essere conformati da oggetti singoli, oppure da altri oggetti composti. Nello sviluppo di software che utilizza un linguaggio Object Oriented, come Java, può succedere di trovarsi di fronte alla necessità di gestire gerarchie di oggetti, all'interno delle quali possiamo trovare oggetti semplici e altri che fanno da contenitore ad oggetti più complessi. Il Composite pattern viene utilizzato dove è necessario trattare un gruppo di oggetti in modo simile ad un singolo oggetto. Il pattern ha lo scopo di comporre gli oggetti con struttura ad albero, utile ad esempio per rappresentare una gerarchia. Questo tipo di design pattern rientra tra i pattern Strutturali. Il pattern utilizza una classe che contiene un gruppo di oggetti, la quale fornisce dei metodi per modificare il gruppo di oggetti.
Questo pattern è utile nei casi in cui si vuole:
- Rappresentare gerarchie di oggetti tutto-parte;
- Essere in grado di ignorare le differenze tra oggetti singoli e oggetti composti.

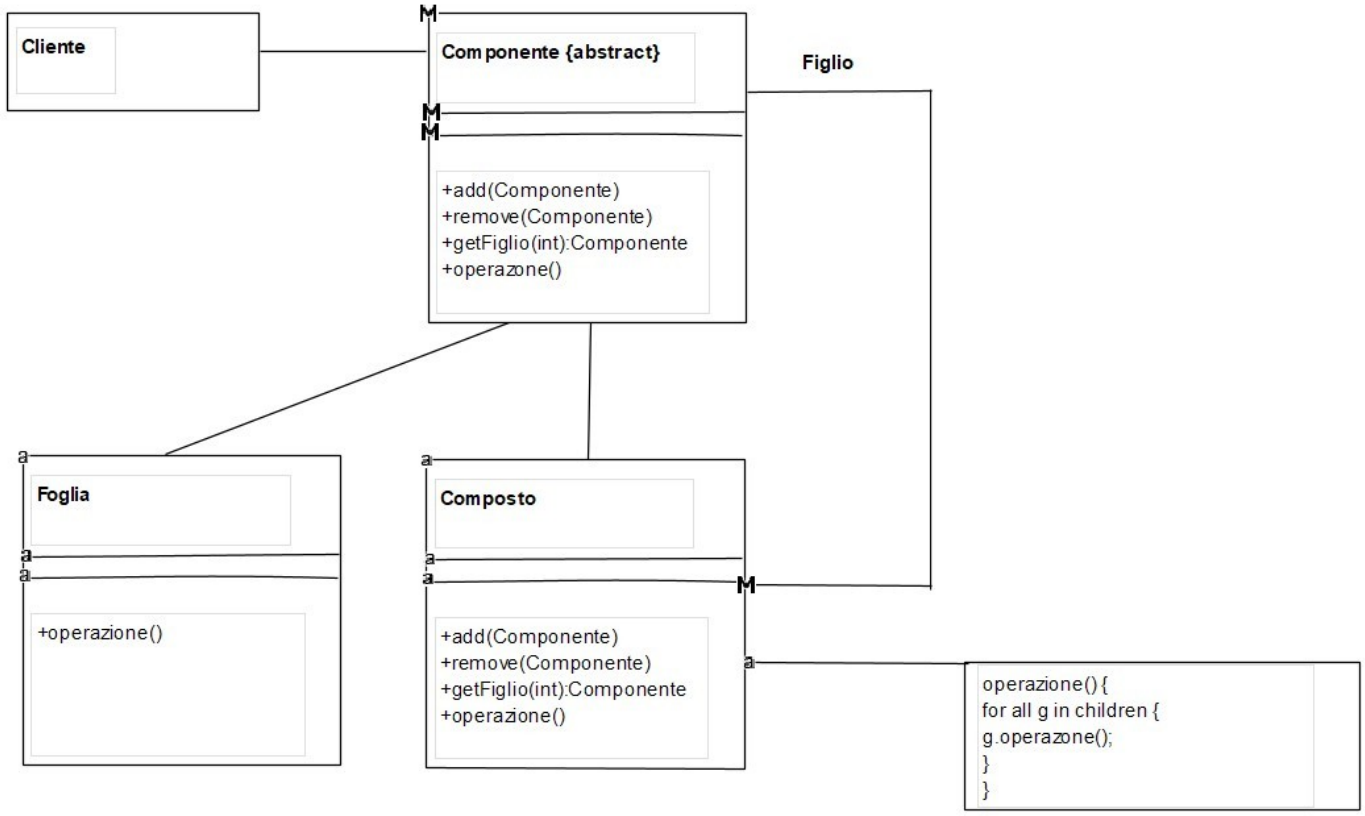
Esempio:
Nel magazzino di una ditta fornitrice di computer ci sono diversi prodotti, quali computer pronti per la consegna, e pezzi di ricambio (o pezzi destinati alla costruzione di nuovi computer). Dal punto di vista della gestione del magazzino, alcuni di questi pezzi sono pezzi singoli (indivisibili), altri sono pezzi composti da altri pezzi. Ad esempio, il "monitor", la "tastiera" e la "RAM" sono pezzi singoli, intanto il "main system", è un pezzo composto da tre pezzi singoli ("processore", "disco rigido" e "RAM"). Un altro esempio di pezzo composto è il "computer", che si compone di un pezzo composto ("main system"), e due pezzi singoli ("monitor" e "tastiera").



Il problema è la rappresentazione omogenea di tutti gli elementi presenti del magazzino, sia dei singoli Componenti, sia di quelli composti da altri Componenti.

- Computer;
- Main System - Monitor – Tastiera;
- Processor - Hard Disk – RAM.

Descrizione della soluzione offerta dal pattern:
Il pattern "Composite" definisce la classe astratta Componente che deve essere estesa in due sottoclassi: una che rappresenta i singoli Componente (Foglia), e un'altra (Composto) che rappresenta i Componenti composti, e che si implementa come contenitore di Componenti. Il fatto che quest'ultima sia un contenitore di Componenti, gli consente di immagazzinare al suo interno, sia Componenti singoli, sia altri contenitori (dato che entrambi sono stati dichiarati come sottoclassi di Componente).



Partecipanti:

Componente: classe astratta Componente.
- Dichiara una interfaccia comune per oggetti singoli e composti.
- Implementa le operazioni di default o comuni tutte le classi.

Foglia: classe ParteSingola.
- Estende la classe Componente, per rappresentare gli oggetti che non sono composti (foglie).
- Implementa le operazioni per questi oggetti.

Composite: classe ParteComposta.
- Estende la classe Componente, per rappresentare gli oggetti che sono composti.
- Immagazzina al suo interno i propri Componenti.
- Implementa le operazioni proprie degli oggetti composti, e particolarmente quelle che riguardano la gestione dei propri Componenti.

Client: in questo esempio sarà il programma principale quello che farà le veci di cliente.
- Utilizza gli oggetti singoli e composti tramite l'interfaccia rappresentata dalla classe astratta Componente.

Descrizione del codice:
La classe astratta Componente definisce l'interfaccia comune di oggetti singoli e composti, e implementa le loro operazioni di default.
Particolarmente le operazioni add(Componente c) e remove(Componente c) sollevano una eccezione del tipo ParteSingolaException se vengono invocate su un oggetto foglia (tentativo di aggiungere o rimuovere un Componente). Invece nel caso di getFiglio(int n), che serve a restituire il Componente di indice n, l'operazione di default restituisce null. Il metodo descrivi() è dichiarato come metodo astratto, da implementare in modo particolare nelle sottoclassi. Il Costruttore di Componente riceve una stringa contenente il nome del Componente, che verrà assegnato ad ognuno di essi.
La classe ParteSingola estende la classe Component. Possiede un costruttore che consente l'assegnazione del nome del singolo pezzo, il quale verrà immagazzinato tramite l'invocazione al costruttore della superclasse. La classe ParteSingola fornisce, anche, l'implementazione del metodo descrivi().

La classe ParteComposta estende anche Componente, e implementa sia i metodi di gestione dei componenti (add, remove, getFiglio), sia il metodo descrivi(). Si noti che il metodo descrivi() stampa in primo luogo il proprio nome dell'oggetto, e poi scandisce l'elenco dei suoi componenti, invocando il metodo descrivi() di ognuno di essi. Il risultato sarà che insieme alla stampa del proprio nome dell'oggetto composto, verranno anche stampati i nomi dei componenti.

Si noti che in questa implementazione ogni ParteComposta gestisce i propri Componente in un Vector.
La classe ParteSingolaException rappresenta l'eccezione che verrà sollevata nel caso che le operazioni di gestione dei Componente vengano invocate su una parte singola.

CompositeClient fa le veci del Client che gestisce i diversi tipi di pezzi, tramite l'interfaccia comune fornita dalla classe Componente. Nella prima parte dell'esecuzione si creano dei pezzi singoli (monitor, tastiera, processore , ram e hardDisk), dopodiché viene creato un oggetto composto (main System) con tre di questi oggetti singoli. L'oggetto composto appena creato serve, a sua volta, per creare, insieme ad altri pezzi singoli, un nuovo oggetto composto (computer). L'applicazione invoca poi il metodo descrivi() su un oggetto singolo, sull'oggetto composto soltanto da pezzi singoli, e sull'oggetto composto da pezzi singoli e pezzi composti. Finalmente fa un tentativo di aggiungere un Componente ad un oggetto corrispondente a un pezzo singolo.

Osservazioni sull'esempio
Si noti che nell'esempio presentato, la classe astratta Componente fornisce un'implementazione di default per i metodi di gestione dei Componente (add, remove, getFiglio). Dal punto di vista del Composite pattern, sarebbe anche valida la dichiarazione di questi metodi come metodi astratti, lasciando l'implementazione alle classi ParteSingola e ParteComposta.
Se si implementa il pattern in questo modo, si devono modificare le classi Componente e ParteSingola. In particolare, il codice della classe Componente dovrebbe dichiarare i metodi di gestione dei Componente (add, remove e getFiglio), come metodi astratti.