Appendice A:

Pattern GoF (Gang of Four) utilizzati

1. Composite

1.1 Introduzione

A seguito di un feedback del Cliente al software prodotto nell'iterazione 3 è emerso un nuovo requisito funzionale che l'applicazione deve fornire e soddisfare.

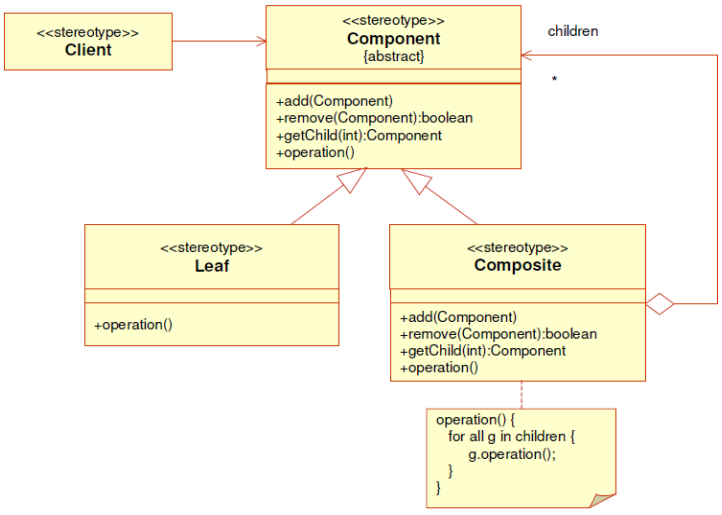
Il Cliente ha richiesto la possibilità di creare e inserire nei Tipi porta dei kit di componenti base. Questi kit devono essere trattati come dei componenti veri e propri e al suo interno possono contenere a loro volta altri kit. Il Cliente esige inoltre che non si possano creare kit di Lamiera.

Il team di Sviluppo dell'applicazione GASH ha pensato di utilizzare il pattern Composite.

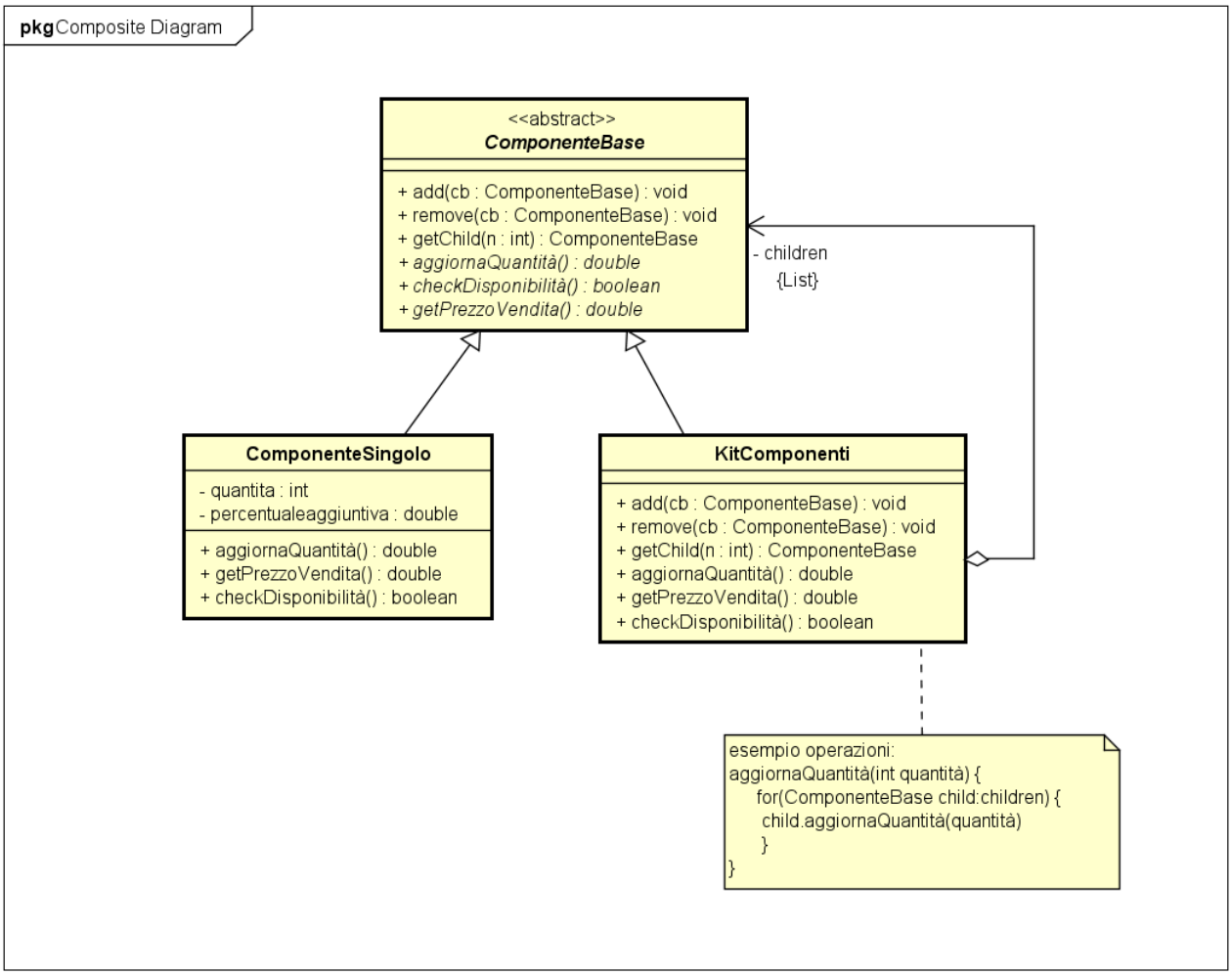
Questo pattern permette di trattare oggetti singoli e composizioni di oggetti in modo uniforme e rappresentare quindi gerarchie di oggetti tutto-parte.

Il pattern Composite definisce diverse classi:

- Una classe astratta "Component", che dev'essere estesa da due sottoclassi, "Leaf" e "Composite".
- Una sottoclasse "Leaf", che rappresenta i singoli componenti (nel nostro caso la classe ComponenteBase, che verrà chiamata diversamente).
- L'altra, "Composite", che rappresenta i componenti composti e si implementa come contenitore di componenti.



1.2 Applicazione del Pattern



ComponenteBase rappresenta la classe "Component" del pattern, quindi è abstract ed estende Componente.

ComponenteBase, come previsto dal pattern, è superclasse di due sottoclassi:

- KitComponente, che rappresenta la classe "Composite" del pattern.
- ComponenteSingolo, che rappresenta la classe "Leaf" del pattern.

La Classe ComponenteSingolo sostituisce la classe precedente ComponenteBase della precedente implementazione.

2. Singleton

Questo pattern ci assicura che una classe abbia un'unica istanza e un punto di accesso globale.

Il team di sviluppo di GASH ha pensato di rendere "Singleton" le classi Gash e Catalogo, in modo tale da impedire la creazione di più istanze e consentire l'accesso all'unica istanza tramite un metodo static getInstance().

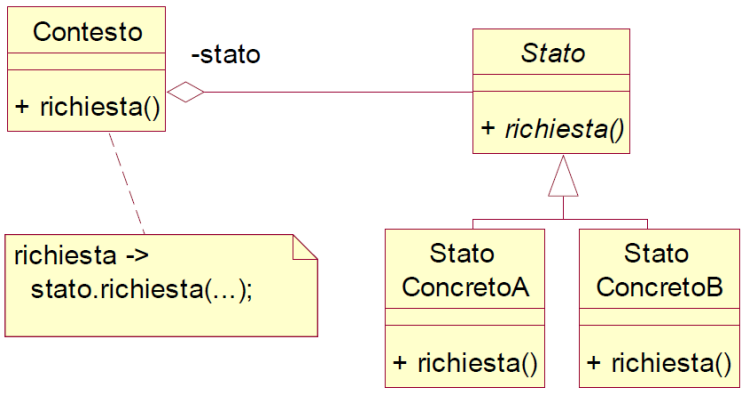
3. State

3.1 Introduzione

L'esigenza di applicare questo pattern nasce in due diverse fasi del progetto: nel refactoring dell'iterazione 2, per tenere traccia dello stato di un ordine, e durante la progettazione dell'iterazione 4, per tracciare lo stato delle consegne.

Il pattern State è un design pattern di tipo comportamentale che permette a un oggetto di cambiare il suo comportamento al variare del suo stato interno.

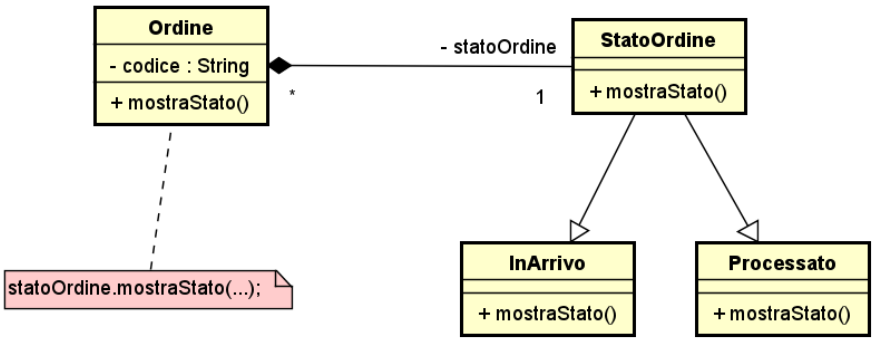
Struttura:



- Contesto: definisce la classe del client e mantiene un riferimento ad uno StatoConcreto.
- Stato: definisce l'interfaccia, implementata dalle classi StatoConcreto, che incapsula la logica del comportamento associato ad un determinato stato.
- StatoConcreto: Implementa il comportamento associato ad un particolare stato.

In questo modo, la logica che implementa il cambiamento di stato viene implementata in una sola classe (*Contesto*) piuttosto che con istruzioni condizionali (*if* o *switch*) nella classe che implementa il comportamento.

3.2 Applicazione del Pattern



Non appena viene effettuato un nuovo ordine, il suo stato viene impostato in “InArrivo”. Al termine della registrazione dei componenti nel sistema lo stato diventerà “Processato”.

L’applicazione del pattern nel caso d’uso UC8 (Gestisci Consegne) è analoga.