

Design Pattern Command => COMPORTAMENTALE.

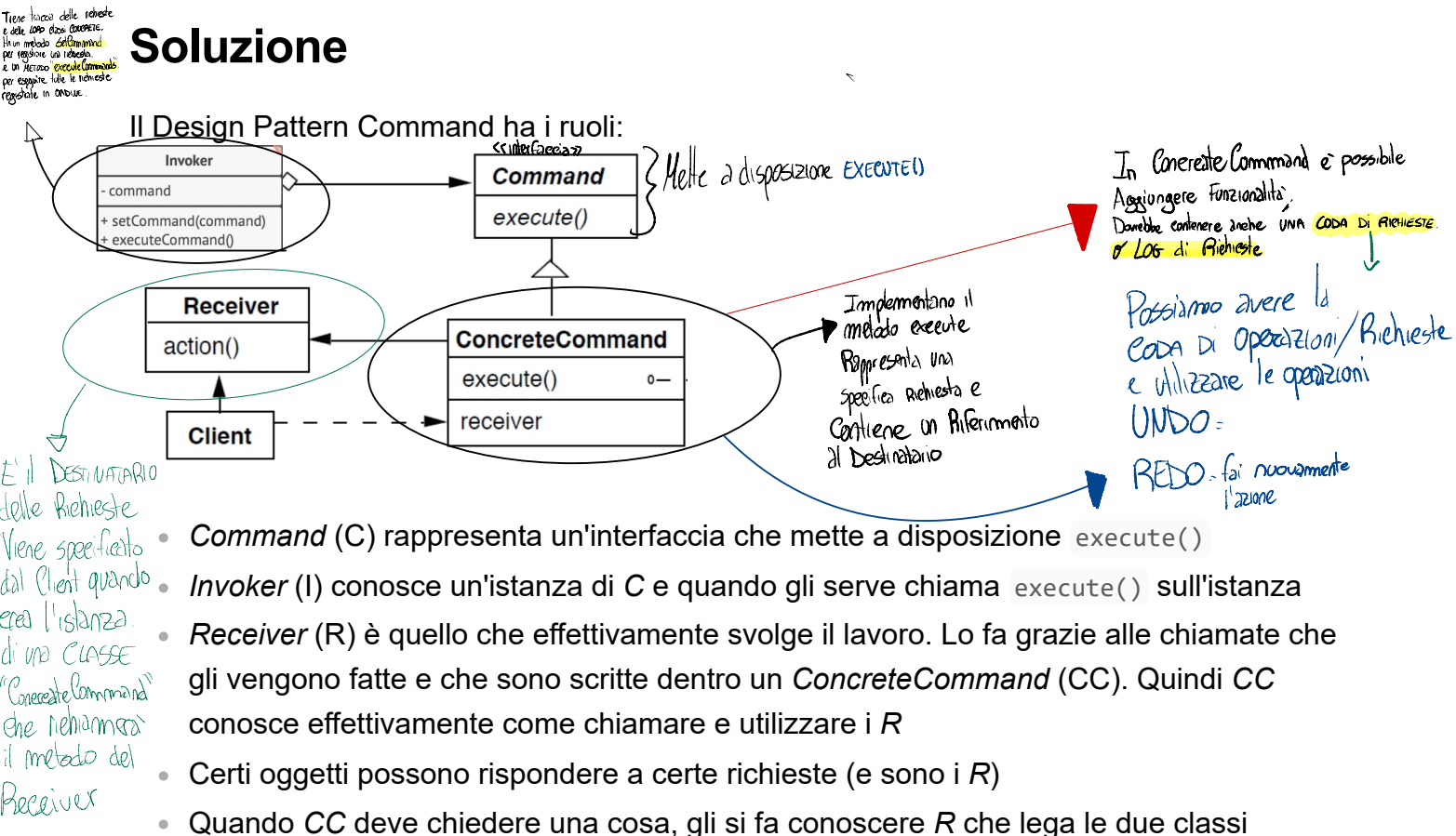
L'intento di questo pattern è di SEPARARE L'emittente (che invia la richiesta) dal DESTINATARIO (che gestisce la richiesta)

Intento Supporta funzionalità di ANNULLAMENTO e la Ripetizione di un'operazione

- Una **richiesta**, di solito, è una **chiamata di metodo**
- La richiesta può diventare un oggetto che incapsula informazioni riguardo a chi potrà portare a termine la richiesta
 - Quindi la richiesta è fatta tramite un oggetto e qualcun altro saprà come portare a termine la richiesta
- Si può **SVINCOLARE** richiedente da chi effettivamente **esegue la richiesta**
- Si può **eliminare la conoscenza dentro il richiedente** così che egli **non conosca chi può ricevere la richiesta** ma si limita a formularla. Quindi prepara un oggetto Richiesta.
- La **richiesta** può essere **passata ad altri oggetti intermedi** in modo da poter essere **manipolata e modificata**, passando quindi per altri oggetti
- Il richiedente prepara un oggetto di tipo **Command** (quindi vi si lega) e tramite l'interfaccia di Command, si possono invocare delle operazioni su questo tipo.
 - Il codice di Command è scritta in modo da sapere come gestire il comando in sé
 - Command, quindi, conosce i dettagli e i modi di operare su un Target ricevente
- Ne segue anche che il **ricevente** deve sapere **come trattare la richiesta** e quindi **come soddisfarla**

Soluzione

Il Design Pattern Command ha i ruoli:



- *Client* (Cl) inizializza un certo attributo *CC* per far usare la giusta istanza di *R*

Collaborazioni

- In *CC* è possibile **aggiungere funzionalità** (tipo: ripristino di stati precedenti. Per es: eliminare l'ultima operazione fatta su un *R*. Cioè una sorta di `unexecute()`)
- *CC* potrebbe anche avere una **coda di richieste o dei log di richieste** (quindi una cronologia di azioni)

Conseguenze

DISACCOPPIAMENTO tra chi manda la richiesta e Receiver
 I dettagli della richiesta sono conosciute dal *ConcreteCommand*.
 È possibile parametrizzare i *CC*. → Per ottimizzare il numero di classi
 (Si CAMBIA i Dati e non si creano altre Classi)

- La parte dell'applicazione che vuole mandare una richiesta è **DISACCOPPIATA** da chi riceverà la richiesta e, inoltre, non conosce tutte le operazioni che effettivamente vengono fatte
- I dettagli della richiesta sono conosciute dal *ConcreteCommand*
- Gli oggetti della gerarchia di *C* possono essere estesi come una normale gerarchia di classi
- I *CC* possono essere anche **composti** (vedi *Composite*) e quindi creando *MacroCommand* che rappresenta una lista di comandi già predisposti
- E' possibile **parametrizzare** i *CC*. Man mano che si cambia quello che c'è all'interno di *CC*, si può cambiare il comportamento totale dell'applicazione. Per questo motivo non è sempre necessario scrivere nuove classi ma basterà **cambiare i parametri**

Implementazione

Si deve scegliere cosa mettere in *CC*:

- potrebbe essere solo un semplice **TRAMITE**, quindi semplice *passaggio di chiamate*. Serve solo per disaccoppiare *Invoker* dal *Receiver*
- potrebbe contenere molti altri controlli e/o metodi per far lavorare *R* in base alla situazione

CC potrebbe sostituirsi interamente a *R* ma è una scelta un po' estrema che degenera e CC ingloba *R*

- Se si volessero usare azioni di **UNDO** e/o **REDO** serve conservare una lista di azioni eseguite

Esempio