

Command

- ▶ **Intento:** incapsulare una richiesta in un oggetto, permettendo quindi di parametrizzare i client con differenti richieste, code o log di richieste, e supportare l'annullamento di operazioni
- ▶ Motivazione
 - ▶ Qualche volta può essere necessario mandare una richiesta a oggetti senza conoscere alcunché dell'operazione richiesta o del ricevente della richiesta
 - ▶ La richiesta viene trasformata in oggetto e quest'oggetto può essere immagazzinato o passato come altri oggetti
 - ▶ La chiave di questo pattern è una classe astratta Command che dichiara un'interfaccia per eseguire operazioni. Le sottoclassi specificano la coppia ricevente e azione, immagazzinando il ricevente come attributo e implementando il metodo che effettua le richieste. Il ricevente conosce ciò che è necessario per servire la richiesta

Prof. Tramontana - Giugno 2023

Command: Collaborazioni

- ▶ Il client crea un oggetto ConcreteCommand e specifica il suo oggetto Receiver
- ▶ Un oggetto Invoker tiene l'oggetto ConcreteCommand
- ▶ L'oggetto Invoker effettua una richiesta chiamando execute() sull'oggetto Command. Quando i comandi sono reversibili, il ConcreteCommand conserva lo stato per cancellare il comando e ritornare a prima della chiamata a execute()
- ▶ L'oggetto ConcreteCommand chiama le operazioni sul suo Receiver per portare a termine la richiesta

Prof. Tramontana - Giugno 2023

Command: Soluzione

- ▶ Struttura
- ▶ Partecipanti
 - ▶ **Command** definisce un'interfaccia per eseguire un'operazione
 - ▶ **ConcreteCommand** definisce il collegamento fra un oggetto Receiver e un'azione; implementa l'operazione execute chiamando le corrispondenti operazioni sul Receiver
 - ▶ **Client** crea un oggetto ConcreteCommand e imposta il suo Receiver
 - ▶ **Invoker** chiede al comando di effettuare la richiesta
 - ▶ **Receiver** conosce come effettuare le operazioni associate alla richiesta. Qualunque classe può essere un Receiver

Prof. Tramontana - Giugno 2023

Command: Conseguenze

- ▶ Il pattern Command disaccoppia l'oggetto che invoca l'operazione da quello che conosce come effettuarla
- ▶ I comandi diventano oggetti e possono essere manipolati ed estesi come altri oggetti
- ▶ Si possono assemblare comandi in un comando composto (MacroCommand). In generale, i comandi composti sono un'istanza del design pattern Composite
- ▶ E' facile aggiungere nuovi comandi, poiché non bisogna cambiare classi esistenti

Prof. Tramontana - Giugno 2023

Command: Implementazione

- ▶ Un Command può avere un'ampia gamma di abilità. Potrebbe solo definire il legame con un ricevente e l'azione da eseguire. Oppure, implementare tutto facendo così a meno del ricevente
- ▶ I Command possono supportare undo se hanno la capacità di rendere reversibile la loro esecuzione. Per questo la classe ConcreteCommand potrebbe aver bisogno di conservare uno stato aggiuntivo. Il ricevente deve fornire le operazioni che permettono al command di far tornare il ricevente allo stato precedente
- ▶ Per un livello di undo il command deve conservare l'ultima attività eseguita, per più livelli di undo (e redo), sarà necessario avere una lista di comandi eseguiti

Command: Esempio

- ▶ Si vuol avere un calendario e un menù per aggiungere o togliere note sul calendario
- ▶ Menu è un Invoker
- ▶ Calendario è un Receiver
- ▶ Updater è un ConcreteCommand

