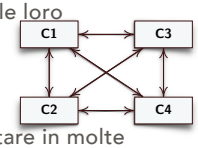
Mediator

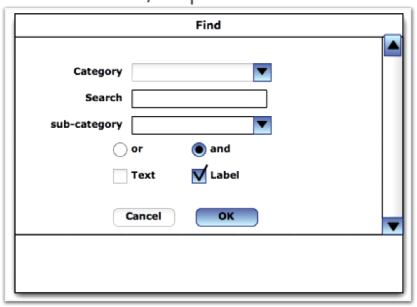
- **Intento**
  - Definire un oggetto che incapsula come un gruppo di oggetti interagisce. Il Mediator promuove il lasco accoppiamento fra oggetti poiché evita che essi interagiscano direttamente, e permette di modificare le loro interazioni indipendentemente da essi

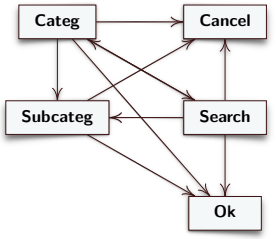


- **Motivazione**
  - La distribuzione di responsabilità fra vari oggetti può risultare in molte connessioni fra oggetti, nel caso peggiore un oggetto conosce tutti gli altri
  - Molte connessioni rendono un oggetto dipendente da altri e l'intero sistema si comporta come se fosse monolitico. Inoltre potrebbe essere difficile cambiare il comportamento del sistema poiché il comportamento è distribuito fra oggetti
  - Si possono evitare questi problemi incapsulando il comportamento collettivo in un oggetto *mediatore* separato. Il mediatore serve da intermediario ed evita che gli oggetti dipendano fra loro

Mediator

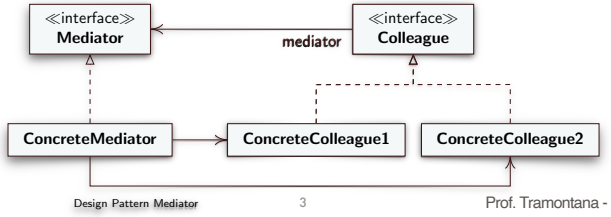
- Per la finestra di ricerca mostrata
  - Ogni elemento visualizzato (testo, lista, bottone, etc.) è controllato da una corrispondente classe
  - Ciascuna classe deve comunicare il suo stato alle altre per far aggiornare la visualizzazione
  - Ciascuna classe (senza un Mediator) chiamerà i metodi di tutte le altre classi, e quindi ciascuna classe è dipendente dalle altre





Mediator

- Applicabilità
  - Usare il Mediator quando
    - Un insieme di oggetti comunicano in modo ben definito ma complesso. Le interdipendenze che ne risultano sono non strutturate e difficili da comprendere
    - Riusare un oggetto è difficile poiché esso comunica con tanti altri oggetti
    - Un comportamento che è distribuito fra tante classi dovrebbe essere modificabile senza dover ricorrere a sottoclassi



Mediator

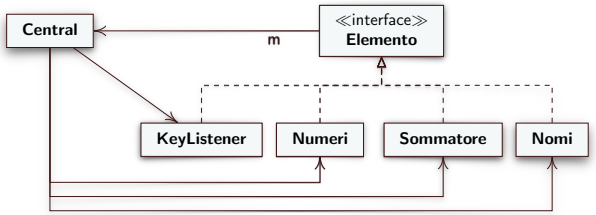
- **Soluzione**
  - Isolare le comunicazioni (complesse) tra oggetti dipendenti creando una classe separata per esse
  - **Mediator** definisce una interfaccia (punto di incontro) per gli oggetti connessi **Colleague**
  - **ConcreteMediator** implementa il comportamento cooperativo e coordina oggetti **Colleague**
  - Ogni **Colleague** conosce il **Mediator**, e comunica con il Mediator quando avrebbe comunicato con un altro **Colleague**
  - I ConcreteColleague mandano richieste, e ricevono richieste, a un oggetto Mediator. Il Mediator implementa il comportamento cooperativo inoltrando le richieste a opportuni ConcreteColleague

Mediator

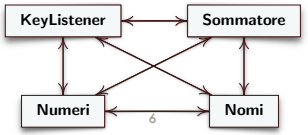
- Conseguenze
- La maggior parte della complessità che risulta nella gestione delle dipendenze è spostata dagli oggetti cooperanti al Mediator. Questo rende gli oggetti più facili da implementare e mantenere
- Le classi Colleague sono più riusabili poiché la loro funzionalità fondamentale non è mischiata con il codice che gestisce le dipendenze
- Il codice del Mediator non è in genere riusabile poiché la gestione delle dipendenze implementata è solo per una specifica applicazione

Esempio

- Si legge un dato dalla tastiera e si compiono delle operazioni, su numeri o su stringa in base al dato letto
- La classe `KeyListener` legge da tastiera un dato e ritorna il valore letto a `Central` (un Mediator), quest'ultima chiama i metodi dei ConcreteCollegue `Numeri`, `Sommatore`, `Nomi`, in base al tipo di dato letto



- Nel caso non si adottasse il Mediator, le varie classi si chiamerebbero fra loro



Esempio

- Il Mediator `Central` avvia la lettura da tastiera tramite il metodo `doWork()` di `KeyListener` e ottiene da esso il valore letto, quindi `Central` chiama `put()` sugli oggetti interessati al valore letto
- Quando un oggetto ConcreteColleague riconosce una condizione di arresto, chiama `stop()` su `Central`, che avvisa gli altri ConcreteColleague
- In figura si mostra il caso in cui `Numeri` chiama `stop()` su `Central`