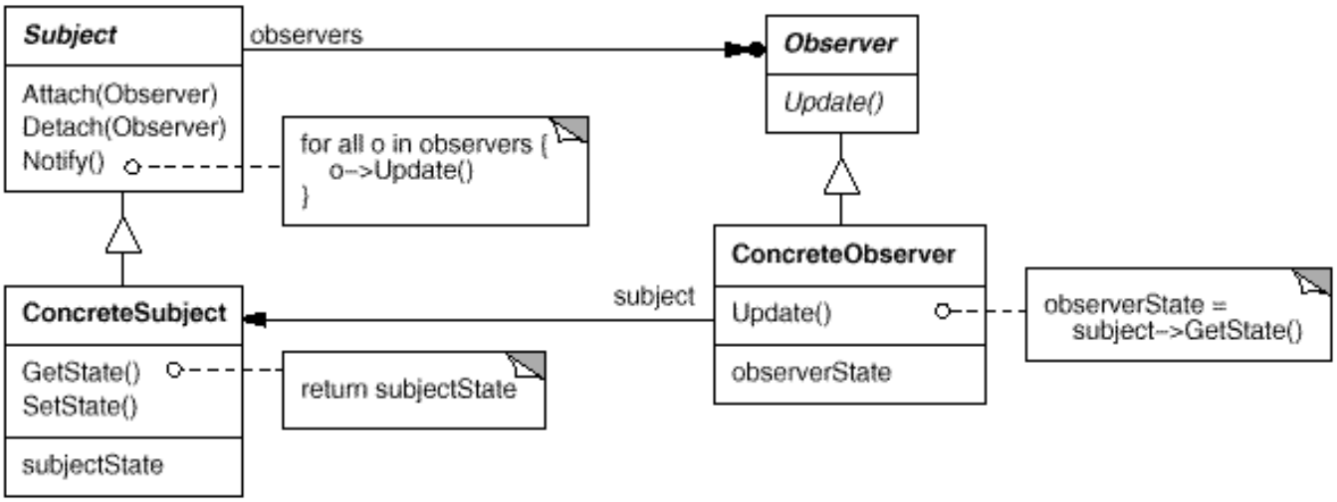
Observer

↓ INTENT ↓

```
Quando la modifica di un oggetto richiede la modifica di altri e non si sa
quanti oggetti devono essere modificati.
```

↓ STRUCTURE ↓

Completo ↓



Java ↓

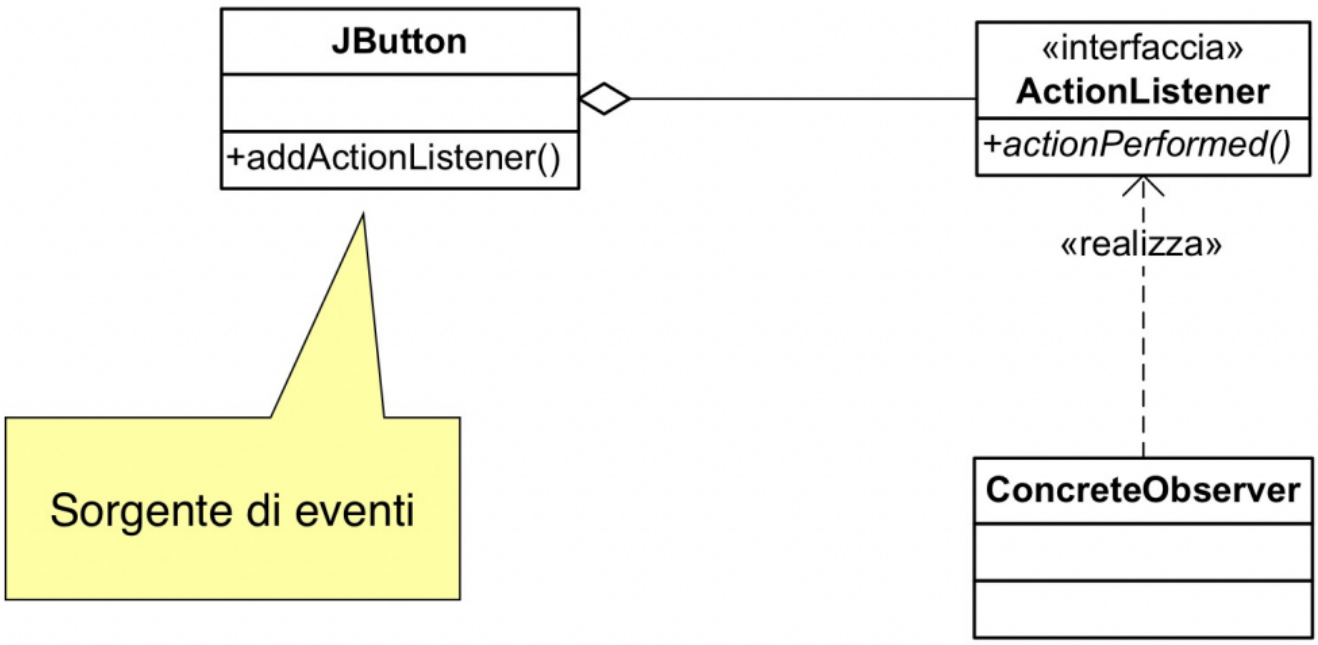
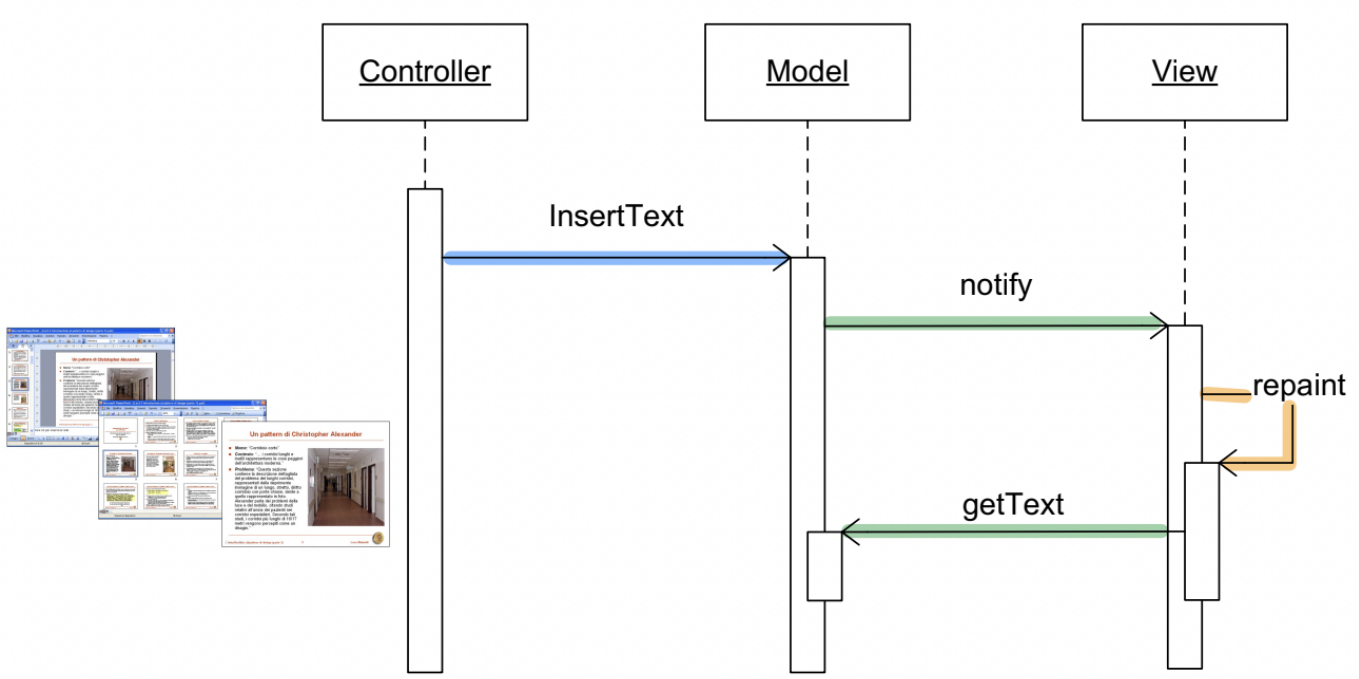


Diagramma temporale ↓



↓ IMPLEMENTATION ↓

---

1. Mappare i soggetti ai loro osservatori.

Tale archiviazione può essere troppo costosa quando ci sono molti soggetti e pochi osservatori.

Si può quindi utilizzare una ricerca associativa (ad esempio una tabella hash) per mantenere la mappatura soggetto-osservatore. Quindi un soggetto senza osservatori non incorre in spese di archiviazione.

---

2. Osservare più di un soggetto.

Potrebbe avere senso in alcune situazioni che un osservatore dipenda da più di un soggetto.

In questi casi è necessario estendere l'interfaccia di aggiornamento per far sapere all'osservatore quale soggetto sta inviando la notifica (passando se stesso come parametro nell'operazione di aggiornamento, facendo sapere all'osservatore quale soggetto esaminare)

---

3. Chi chiama Notify e attiva l'aggiornamento?

- Avere operazioni di impostazione dello stato sulla chiamata del soggetto Notifica dopo aver modificato lo stato del soggetto.
    - **Vantaggio**: i clienti non devono ricordarsi di chiamare Notify sull'argomento.
    - **Svantaggio**: operazioni consecutive causano diversi aggiornamenti consecutivi. Potrebbe essere inefficiente.
- Rendi i clienti responsabili di chiamare Notify al momento giusto.
    - **Vantaggio**: il cliente ha la gestione degli aggiornamenti, evitando così inutili aggiornamenti.

- **Svantaggio**: il cliente ha la responsabilità aggiuntiva di attivare l'aggiornamento, rendemdo più probabili gli errori.

---

4. Riferimenti penzolanti a soggetti cancellati.

Un modo per evitare riferimenti penzolanti è fare in modo che il soggetto informi i suoi osservatori quando viene cancellato in modo che possano reimpostare il loro riferimento ad esso.

La semplice eliminazione degli osservatori non è un'opzione, perché altri oggetti potrebbero fare riferimento a loro o potrebbero osservare anche altri soggetti.

---

5. Assicurarsi che lo stato del soggetto sia coerente prima della notifica.

È importante perché il soggetto è interrogato per il suo stato attuale nel corso dell'aggiornamento del proprio stato. È facile da violare quando le operazioni della sottoclasse Oggetto chiamano operazioni ereditate.

È possibile evitarlo inviando notifiche dai Model in classi di soggetti astratte, fecendo in modo che Nofica sia l'ultima operazione nel Model, che assicurerà che l'oggetto sia autoconsistente quando le sottoclassi sovrascrivono le operazioni del soggetto.

---

6. Evitare protocolli di aggiornamento specifici dell'osservatore: i modelli push e pull.

Le implementazioni del modello Observer spesso fanno trasmettere al soggetto informazioni aggiuntive sul cambiamento. Il soggetto passa queste informazioni come argomento ad Aggiorna. La quantità di informazioni può variare notevolmente.

- **Modello PUSH**: *presuppone che i soggetti sappiano qualcosa sui bisogni dei loro osservatori*. I soggetti forniscono agli osservatori informazioni dettagliate sul cambiamento. (osservatori meno riutilizzabili)
- **Modello PULL**: *enfatizza l'ignoranza del soggetto*. Il soggetto invia la minima notifica e gli osservatori chiedono esplicitamente i dettagli in seguito. (può essere inefficiente)

---

7. Specificazione esplicita delle modifiche di interesse.

È possibile migliorare l'efficienza degli aggiornamenti consentendo la registrazione degli osservatori solo per eventi di interesse specifici. Per fare ciò sono attaccati ai loro soggetti usando

```
    void Attach(Osservatore, interesse);
```

Al momento della notifica, il soggetto fornisce l'aspetto modificato ai suoi osservatori. Per esempio:

```
    void Update(Subject, interesse);
```

---

8. Incapsulamento di semantiche di aggiornamento complesse.

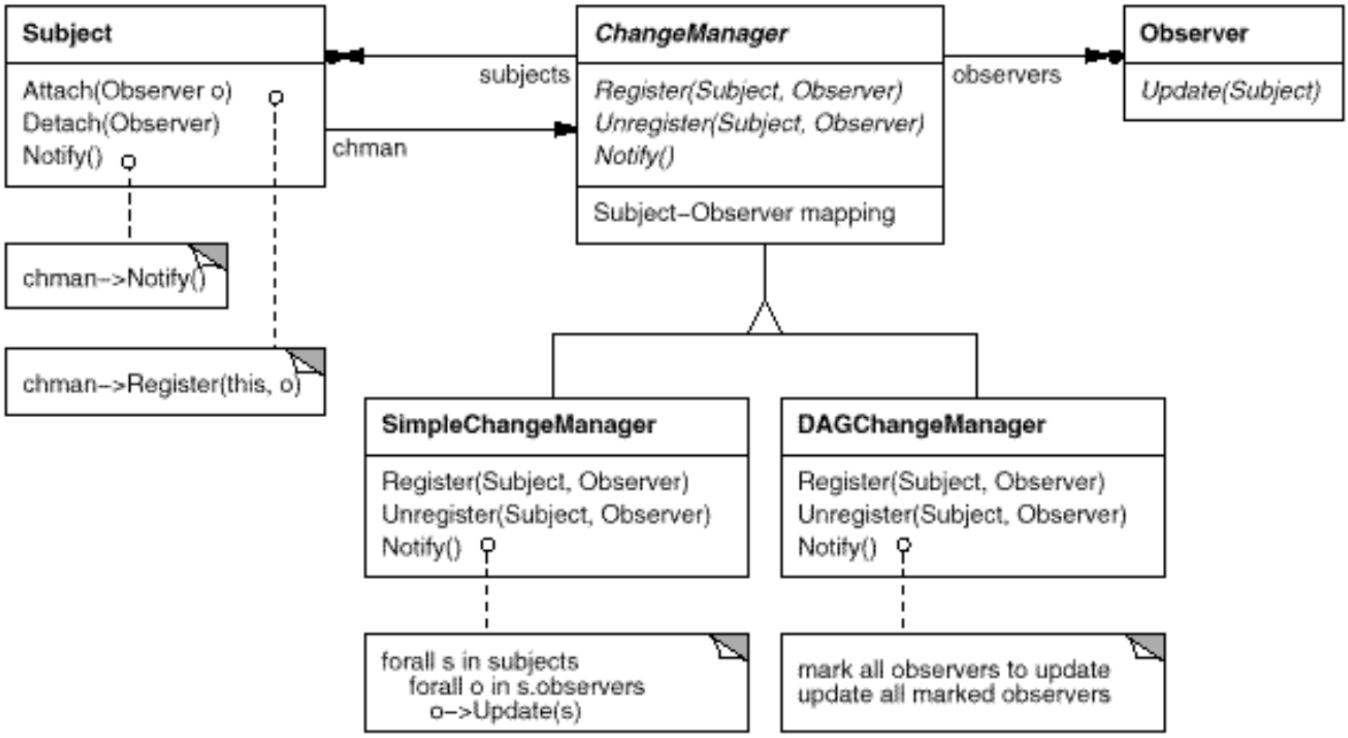
- **ChangeManager**: il suo scopo è ridurre al minimo il lavoro necessario per far riflettere gli osservatori su un cambiamento nel loro soggetto.

Ad esempio, se un'operazione comporta modifiche a più soggetti, potresti dover assicurarti che i loro osservatori vengano avvisati dopo che tutti i soggetti sono stati modificati per evitare di avvisare gli osservatori più di una volta.

ChangeManager ha tre responsabilità:

1. Mappa un soggetto ai suoi osservatori e fornisce un'interfaccia per mantenere questa mappatura. Ciò elimina la necessità per i soggetti di mantenere riferimenti ai propri osservatori e viceversa.
2. Definisce una particolare strategia di aggiornamento.
3. Aggiorna tutti gli osservatori dipendenti su richiesta di un soggetto.

Il DAGChangeManager assicura che l'osservatore riceva un solo aggiornamento. SimpleChangeManager va bene quando più aggiornamenti non sono un problema (il modello Singleton sarebbe utile qui).



↓ EXAMPLE ↓

```
/**
 * Creazione del frame e del pulsante
 */
JFrame f = new JFrame();
JButton b = new JButton("Decliccato");

/**
```

```
 * Si aggiunge al frame il pulsante
 */
f.add(b);

/**
 * Creazione di un Ascoltatore di Azioni
 *  per andare a notificare i cambiamenti alla view
 */
ActionListener al = new ClickListener();

/**    !!! PARTE IMPORTANTE PER L'OBSERVER !!!
 * Si aggiunge al pulsante l'ActionListener
 *  in modo che i due siano collegati
 */
b.addActionListener(al);

/**
 * Setting del frame
 */
f.setSize(300, 200);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
```