# Observer

Publish/Subscribe

MVC



## Observer

- Intento:
  - Definire una **dipendenza uno a molti** fra oggetti
  - quando un oggetto cambia stato tutti i suoi oggetti dipendenti sono notificati e aggiornati automaticamente
- Motivazione:
  - **mantenere la consistenza** fra oggetti che hanno relazioni
- Conosciuto anche come Publish-Subscribe



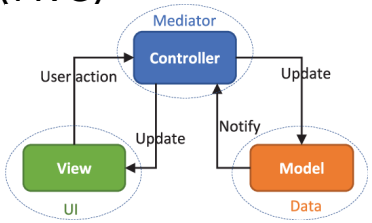
## Observer

- Applicabilità
  - Un'astrazione ha **due aspetti interdipendenti** (es. dati e presentazione). Incapsulare questi aspetti in oggetti separati permette di **riusarli indipendentemente**
  - Un cambiamento su un oggetto richiede il cambiamento di altri e **non si conosce quanti** oggetti sarà necessario cambiare
  - Un oggetto deve notificare altri oggetti **senza sapere chi sono** tali oggetti, quindi gli oggetti **non devono essere strettamente accoppiati**

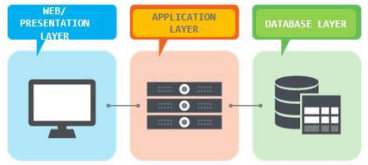


## Pattern Model View Controller (MVC)

- Se non consento interazione diretta tra view e model, rappresenta l'equivalente centralizzato dell'architetturale *three-tier* per sistemi distribuiti

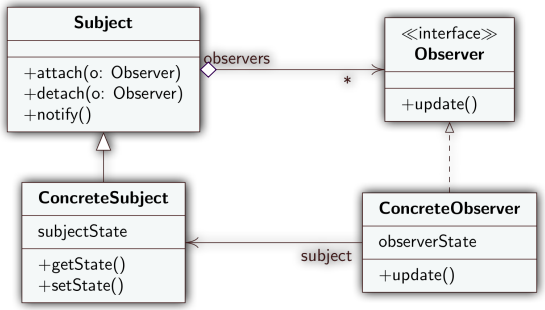



- Vantaggi dovuti alla modularità e disaccoppiamento, ad es. Poter inserire facilmente firewall di protezione tra lato utente (pericoloso) e lato server (intranet)



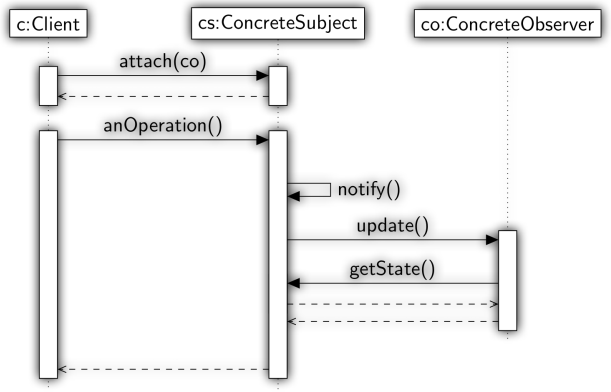
# Observer

- Soluzione
  - Diagramma delle classi



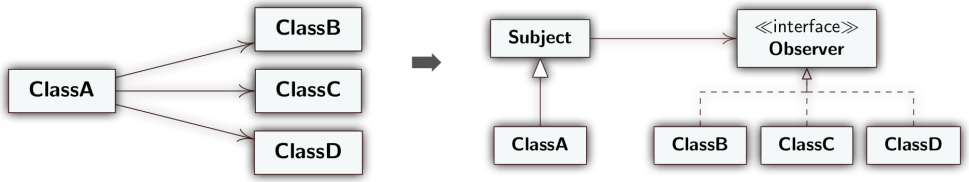



# Observer



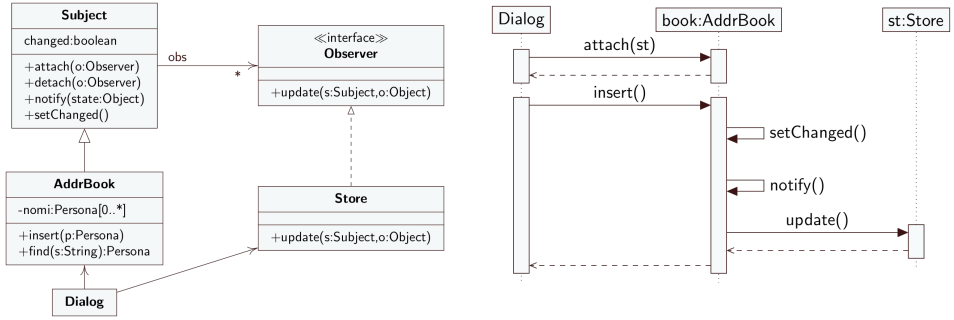



# Prima e dopo l'uso di Observer






# Esempio



# Observer

- Conseguenze
  - Il Subject conosce solo la classe Observer e non ha bisogno di conoscere le classi ConcreteObserver. **ConcreteSubject e ConcreteObserver non sono accoppiati** quindi più facili da riusare e modificare
  - La notifica inviata da un ConcreteSubject è mandata a tutti gli oggetti che si sono iscritti, il ConcreteSubject non sa quanti sono. **Gli osservatori possono essere rimossi in qualunque momento**. L'osservatore **sceglie se gestire o ignorare la notifica**
  - L'aggiornamento da parte del Subject **può far avviare tante operazioni** sugli Observer e altri oggetti per gli aggiornamenti. La notifica può **non dire agli Observer cosa è cambiato** nel ConcreteSubject, è un evento che **indica solo il completamento di un'operazione** del ConcreteSubject



# Considerazioni

- Un Observer potrebbe osservare più oggetti, per sapere chi ha mandato la notifica, come parametro di `update()`, si manda il riferimento al Subject (**relazione N a N**)
- In alcuni casi, per mantenere la consistenza fra Observer e Subject, anche un cambiamento dell'Observer deve essere comunicato al Subject (con **`setState() nel subject`**)
- Quando si vuol **eliminare un Subject**, gli Observer dovrebbero essere avvisati per cancellare il loro riferimento al Subject!
- Il Subject può passare le informazioni quando chiama `notify()`, **modello push**; anziché aspettare che l'Observer legga lo stato, **modello pull**

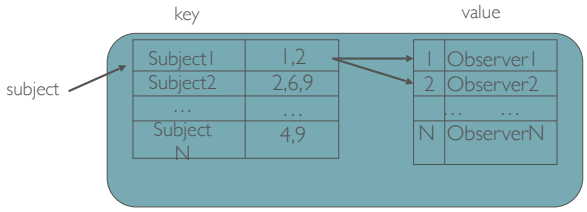


# Considerazioni

- Quando il ConcreteSubject chiama `notify()`, questo chiama `update()` che **esegue sul thread del chiamante**, costringendo il subject ad aspettare l'esecuzione di `update()` di ciascun ConcreteObserver
- Le notifiche **sono in serie** e possono essere **costose,** ovvero tante e lunghe, in modo imprevedibile per il subject!
- Subject chiama `notify()` dopo un cambiamento, oppure si aspetta un certo numero di cambiamenti, in modo da **evitare continue notifiche** agli Observer (**setChanged()** per almeno uno…)
- Gli Observer che si registrano possono **specificare gli eventi di interesse**, in modo che **la notify() non sia un broadcast** e si riducano le update (*Java Event Model*)



# Shared look-up table




- No duplicate references to observers
- No storage for subjects with no observers

# Observer in Java (fino a 8)

- Il pattern Observer è così comune che è stato incluso nella libreria java.util, con i la classe **`Observable`** e l'interfaccia **`Observer`**
- **`Observable`** svolge il ruolo di Subject e notifica il cambiamento di stato quando il metodo ***`notifyObservers`***`()` viene chiamato
- La **classe `Observable`** ha una variabile (flag) di stato, impostata dal metodo ***`setChanged`***`()`.
- **Observer è una interfaccia** che ha solo il metodo *`update`*`()` e può avere un argomento che indica quale oggetto ha causato l'aggiornamento
- Alla base dell'implementazione della gestione eventi (ActionListener) nelle librerie AWT Java 7
- **Seriale e Obsoleto!**

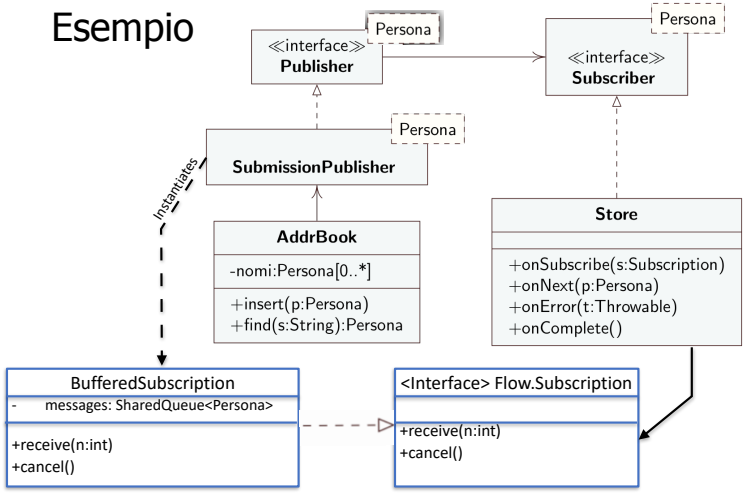


# Observer in Java 9: Publisher/Subscriber

- si è cercato di risolvere il problema del passaggio di un insieme di item da un **publisher** a un **subscriber** **senza bloccare il publisher** e **senza inondare il subscriber**
- Java 9 fornisce nella libreria java.util.concurrent le interfacce **`Publisher`**<T>, **`Subscriber`**<T>, **`Subscription`**, e la classe **`SubmissionPublisher`**. Quest'ultima implementa un Publisher dedicando un thread per mandare ciascun messaggio ai Subscriber



# Esempio






# esempio SubmissionPublisher

- Si ha la concorrenza nelle notifiche, quando ho più soggetti e più osservatori
- Si rende più lasco il legame tra subject e oggetto delegato a smistare le notifiche ai soggetti (SubmissionPublisher)
- **Subscription:** incapsula il collegamento Publisher/Subscriber, fornisce **request() e cancel() e rappresenta un corrispondente submissionPublisher**
- posso avere serie di notifiche distinte per tipologia, e ogni osservatore può registrarsi come interessato solo ad alcune di esse
- Posso di avere più sottoscrizioni per lo stesso osservatore e gestirle in modo differenziato