Design pattern Adapter (object adapter e class adapter)

Questo è un design patter **STRUTTURALE**. I precedenti visti erano **CREAZIONALI**.

Intento

Converte l'interfaccia in modo da adattarla al tipo che si aspetta il client.

Converte l'interfaccia di una classe in un altra interfaccia che si aspetta il client. Il client si aspetta una certa interfaccia (un certo nome di metodo con un certo tipo di parametro in ingresso). Quindi c'è **INCOMPATIBILITA'** fra la richiesta del client e l'effettiva implementazione.

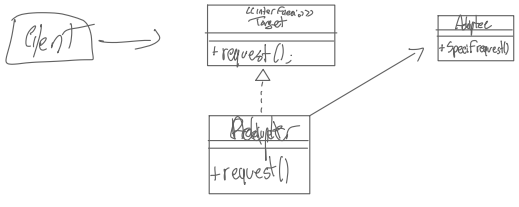
Problema

(Abbiamo un interfaccia che non va bene al client, NON POSSIAMO MODIFICARE METODI) IL CHIAMANTE NON PUÒ ESSERE MODIFICATO.

Si è progettato un sistema software facendo riferimento a una certa interfaccia da usare.

> Si è implementato, per esempio, un metodo getCosto() ma effettivamente serve un getCosto(`parametro`).

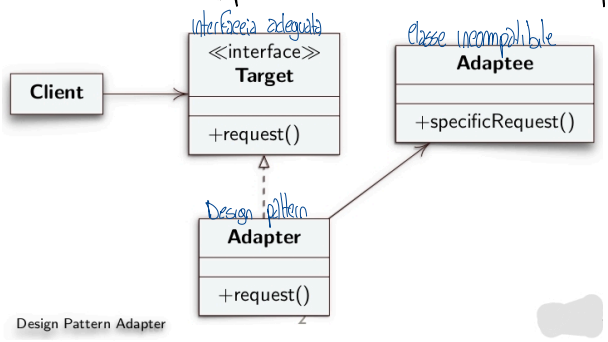
- La parte che deve fornire il risultato(il chiamante) **NON può essere modificata**.
- Cambiare l'interfaccia in modo da adattarsi ai requisiti potrebbe creare problemi se tali metodi sono usati anche altrove.
  - **Se modifico un metodo**, questo stesso deve venire modificato in modo da attenersi all'attività che deve svolgere.



Esempio

- *Alcune volte una classe di una libreria non può essere usata poiché incompatibile con l'interfaccia che si aspetta l'applicazione. Ovvero nome metodo, parametri, tipo parametri di chiamate all'interno dell'applicazione non sono corrispondenti a quelli offerti da una classe di libreria.*
- *Inoltre non è possibile cambiare l'interfaccia della libreria, poiché non si ha il sorgente (comunque non conviene cambiarla*

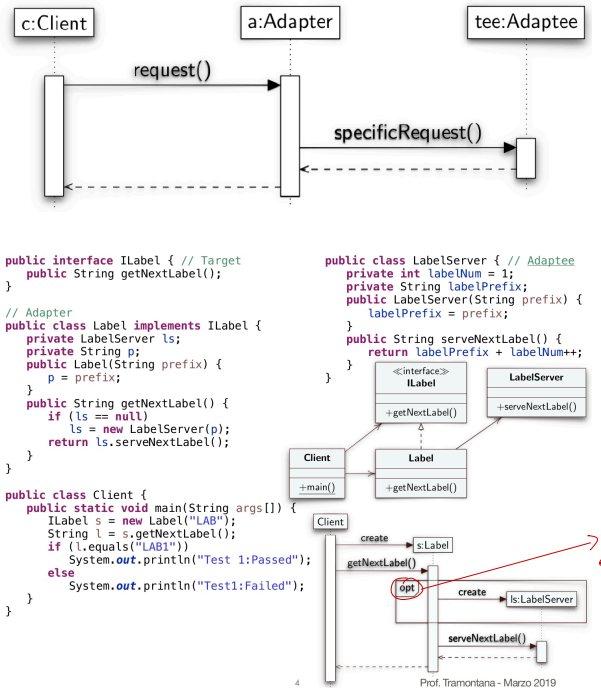
Soluzione



Crea una nuova classe Adapter che farà da adattatore fra invocazioni del client e server.

- I metodi saranno **conformi al client** (*chiamante*).
- La classe che mette a disposizione i metodi saranno conosciuti all'interno della classe Adapter. Quindi da Adapter si chiamano i metodi effettivi.
- Il client si interfaccia con **Target** che è la classe che si aspetta
- Adaptee sarà la classe di libreria che mette a disposizione il servizio che risulta **INCOMPATIBILE** con il Client.

(ESAME): Pronunciare bene il nome delle classi per evitare fraintendimenti. -> *Adapter* e Adaptee (*Adaptìì*).



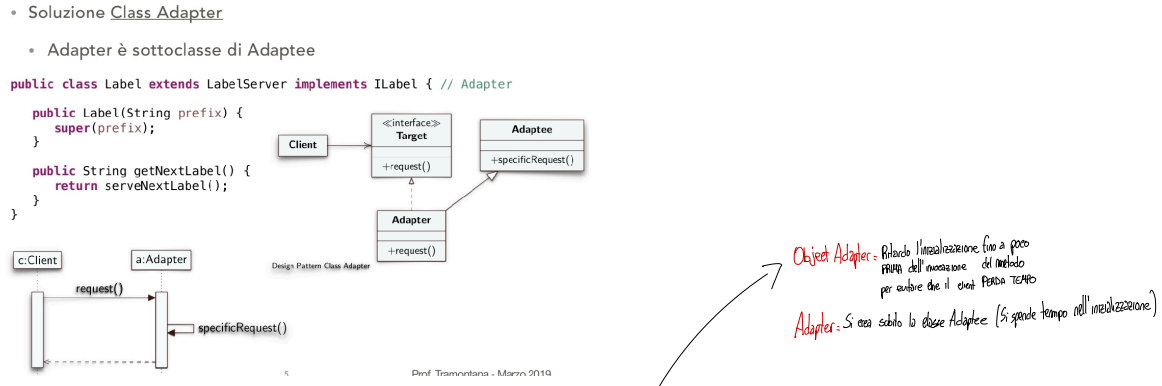
> Si crea **un Adapter per ogni classe che devo adattare**.
> *Se ho **2 librerie** da adattare allora ho 2 **Adapter diversi**.*

- `opt` sta a significare che, quando si deve istanziare `ls` non è detto che essa venga fatta perchè c'è un'istruzione condizionale e quindi potrebbe non venire fatta.

- Questo è un **Object Adapter** perchè si ha **l'invocazione fra oggetti**

Variante Class Adapter

- Adapter è sottoclasse di Adaptee.
- Adapter implementa l'interfaccia Target (*come prima*)
- Si possono liberamente invocare i metodi della superclasse e quindi non serve più un riferimento ad `Adaptee` (un oggetto `Adaptee`)
- Quando devo invocare un metodo di Adaptee, non serve più l'istanza ma invoco il metodo sulla **stessa istanza**



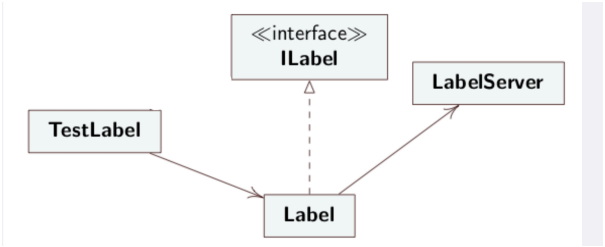
Differenze Object Adapter e Class Adapter

- Dipendono dai requisiti, ovviamente, e dalle scelte di progettazione
- si crea un'unica istanza di Adapter che **contiene tutto**. Se la superclasse richiede molta memoria allora si inizializza anche Adaptee allo stesso momento e quindi si spende quel tempo di inizializzazione (*a prescindere*)
- In Object Adapter, invece, **rimando l'inizializzazione fino a poco prima dell'invocazione del metodo**. (quindi il più tardi possibile, in modo da non far perdere tempo al Client)
  - Adaptee potrebbe non essere mai inizializzato (*in base all'uso del software*) e quindi si **risparmia tempo**. Questo *modo di fare* è detto **LAZY INITIALISATION** (cioè **inizializzazione in ritardo, pigra**)
- **Ci sono MOMENTI DIVERSI di inizializzazione**

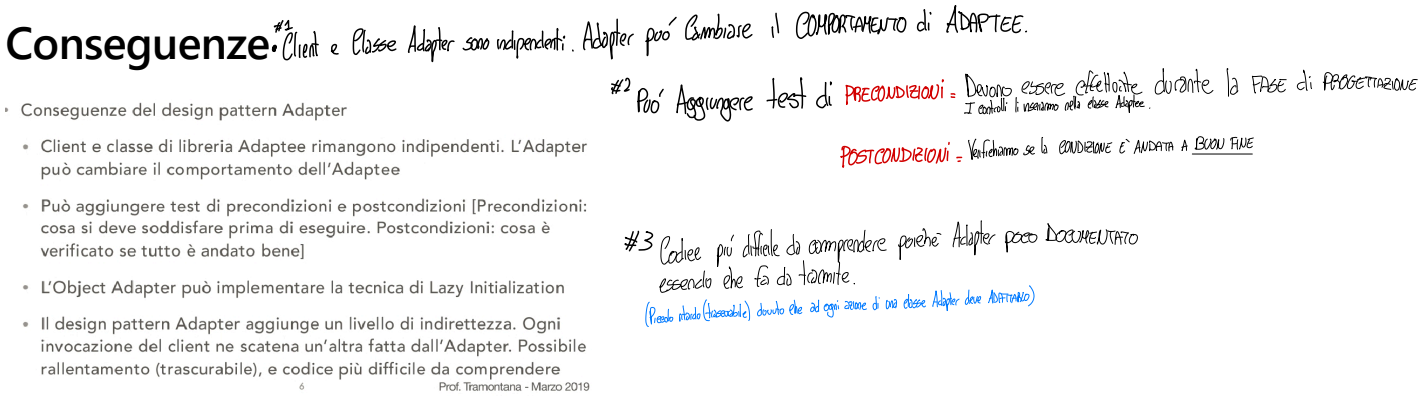
> In Java non è concessa l'ereditarietà multipla per semplificare la scrittura del codice e per evitare incompatibilità fra metodi ereditati da una classe `A` o `B` (**aventi la stessa *signature***)
> **Si può, invece, ereditare da una classe A e da un'interfaccia B**.

Variante Adapter a due vie Utile per far comunicare due classi incompatibili

Fornisce sia l'interfaccia di Target e sia l'intefaccia di Adaptee. Se un client si riferisce a due interfacce. Usando il object adapter uso questa versione e al suo interno istanzio Adaptee (*LabelServer*)



- Il ***Client*** deve invocare i **metodi dell'intefaccia *Target* e i metodi dell'interfaccia *Adaptee***
- *Client* si interfaccia con *Adapter* essendo più "ricca" di informazioni rispetto a *Target*



- **PRECONDIZIONE**: deve verificarsi prima dell'invocazione di un metodo per consentire al metodo di essere **BEN ESEGUITO**.
  - Se queste precondizioni possono essere scritte, **evito** di fare chiamate di metodo che molto **probabilmente incontra errori di esecuzione** o non ritorna il dato voluto.
  - Le precondizioni devono essere individuate durante la progettazione. Vengono messe nel Adapter per non farlo fare al Client.
  - Se si vuole semplificare l'Adaptee, tali controlli si fanno sempre nell'Adapter
- **POSTCONDIZIONI**: si devono verificare **se la condizione è andata a BUON FINE**.
  - *Per esempio, **un valore di ritorno** non deve essere mai nullo oppure deve essere di un certo tipo.*

L'Adapter introduce un livello di dipendenza: ogni volta che viene usato, esso chiama un metodo da Adaptee. Non si passa da chiamante->chiamato ma si aggiunge qualcosa **in mezzo**. In questo caso, a livello di leggibilità di codice, essa viene meno perchè magari Adapter non è nemmeno documentato perchè fa solo da "*tramite*".