Quali sono gli svantaggi?

- Il cliente sta DAVVERO diventando più complesso? Nel nostro primo esempio, il codice paperboy che utilizzava il Cliente doveva anche conoscere e manipolare il Portafoglio. Non è più così...
- Se il cliente è diventato PIÙ complesso, allora perché i clienti sono ora MENO complessi? Esiste ancora la stessa complessità, è solo meglio contenuta nell'ambito di ogni oggetto ed esposta in modo sano.
- Contenendo questa complessità, otteniamo tutti i vantaggi discussi sopra. Quindi sì, possiamo ammettere che il cliente è diventato più complesso. La verità è che si tratta di una complessità che esisteva prima nel codice, e potrebbe essere esistita in diversi punti. Ora si verifica una sola volta e può essere mantenuto nello stesso punto in cui si verificano le modifiche che possono interromperlo.

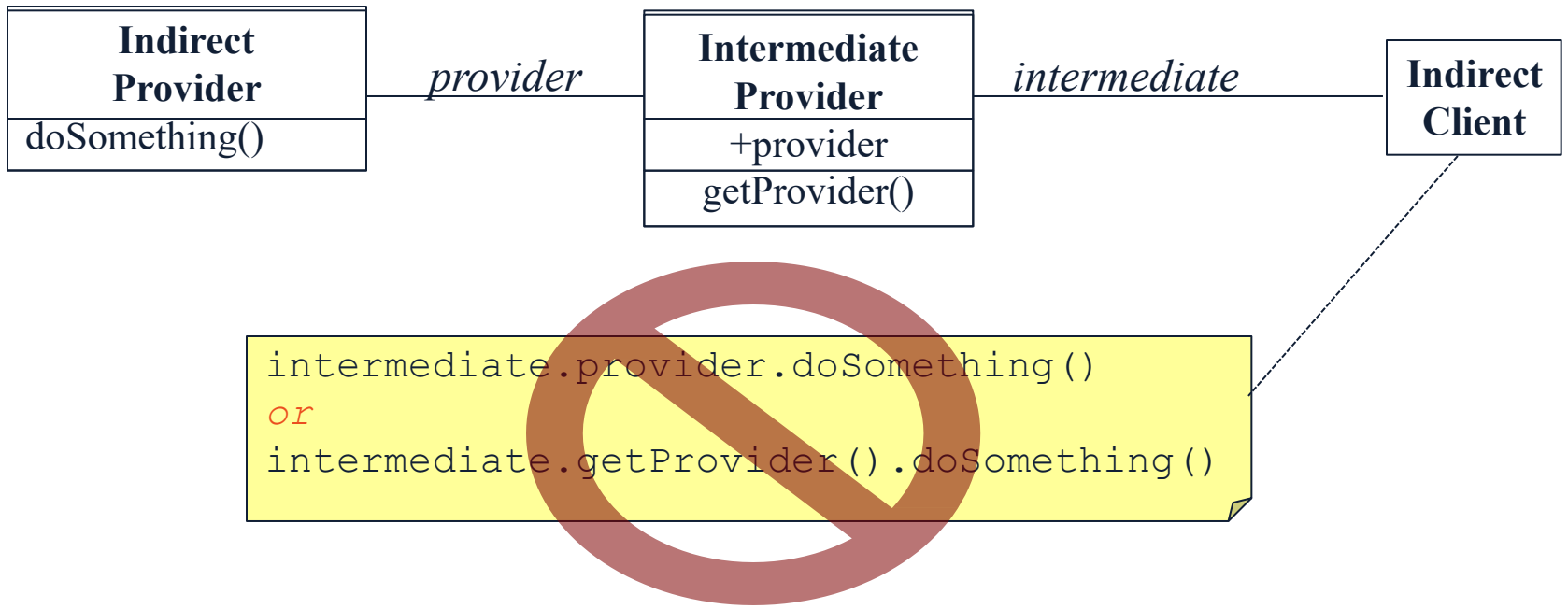
La Legge di Demetra

- E’ un’altra linea guida chiave per avere Basso Accoppiamento
- Utilizzata nel progetto Demetra, uno dei primi grossi sistemi O-O, sviluppato all’univ. di Boston.
- Concetto di base: Spingere al massimo l'information hiding
  - No situazioni tipo: Foo.getA().getB().getC()…..
  - Più è lungo il percorso di chiamate attraversato dal programma, più esso è fragile a cambiamenti

La Legge di Demetra

- Un metodo dovrebbe mandare messaggi solo a:
  - L’oggetto contenente (this)
  - Una variabile di istanza di this
  - Un parametro del metodo
  - Un oggetto creato all’interno del metodo

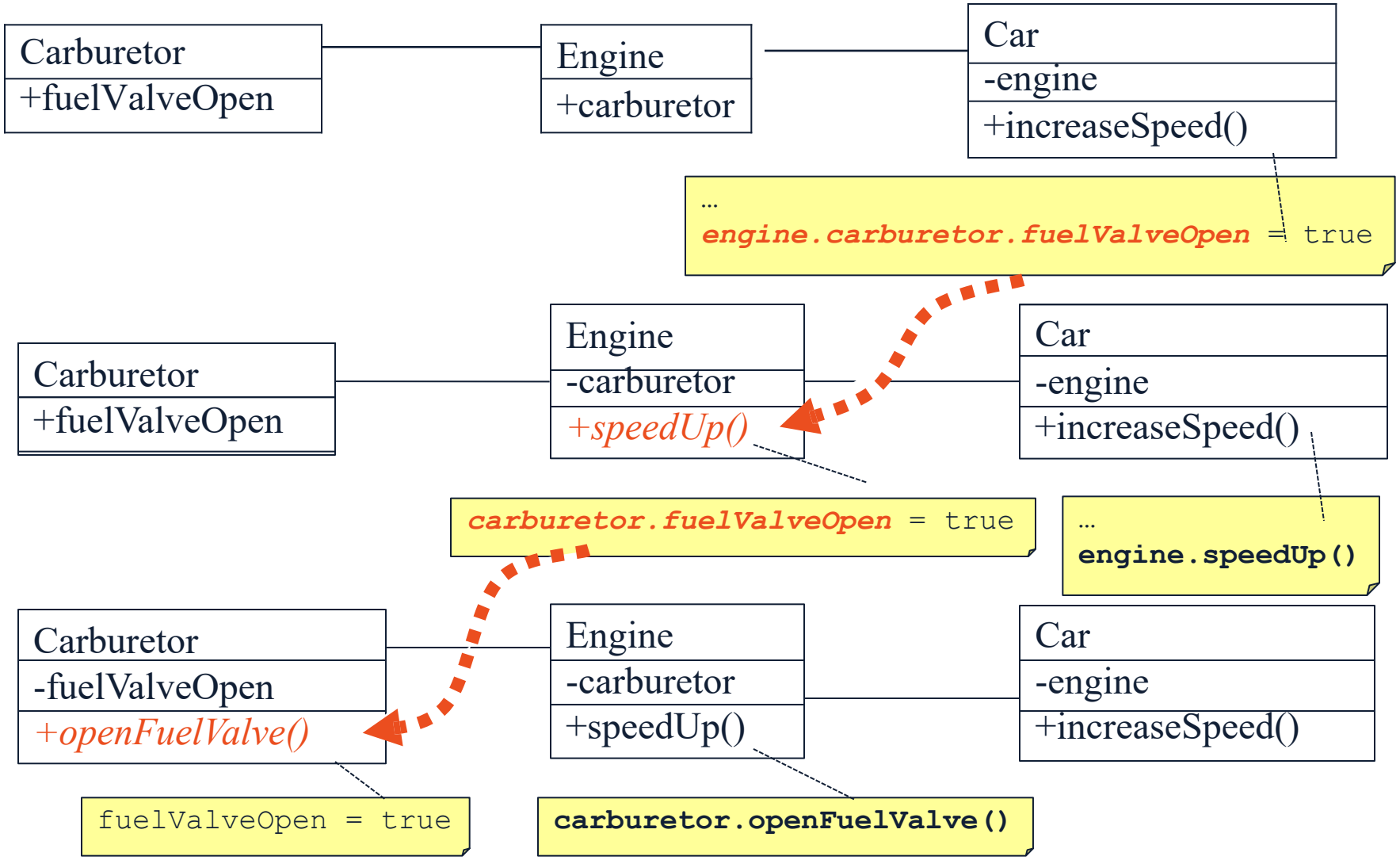
The Law of Demeter, violated



Law of Demeter: A method "M" of an object "O" should invoke only the methods of the following kinds of objects.

1. itself
2. its parameters
3. any object it creates /instantiates
4. its direct component objects

Eliminate Navigation Code



Responsibility-driven design

- Pensare alla progettazione di un sistema in termini di:
  - Classe
  - Responsabilità
  - Collaborazioni
- Ciascuna classe dovrebbe avere delle responsabilità e dei ruoli ben precisi→ Alta Coesione
- La classe che possiede i dati dovrebbe essere responsabile di processarli → Basso Accoppiamento
  - Ciascuna classe dovrebbe essere responsabile di gestire i propri dati → Incapsulamento
- E' facilitata dall'uso di CRC Cards

Le CRC Cards

- Introdotte Kent Beck e Ward Cunningham per insegnare progettazione O-O
- Una card CRC è una scheda da associare ad ogni classe, per rappresentare i seguenti concetti:
  - Nome della Classe
  - Responsabilità della Classe
  - Collaboratori della Classe

- Responsabilità: conoscenza che la classe mantiene o servizi che la classe fornisce
- Collaboratore: un’altra classe di cui è necessario sfruttare la conoscenza o i servizi, per ottemperare alle rensponsabilità sopra indicate

Una CRC Card

- Associata ad ogni classe del class diagram di design

Class Name	
Superclasses	
Subclasses	
Responsibilities	Collaborators

Conseguenze

- Le CRC cards spingono ad ottenere un design con chiare responsabilità e controllo sul numero di classi associate
  - Se non riesco a specificare chiaramente le responsabilità di una classe, c'è un errore di design
  - Se una classe ha bisogno di troppe classi collaboratrici, c'è un potenziale errore di design

Localizzare le modifiche

- La progettazione RDD, con CRC Cards, porta ad una localizzazione delle modifiche
- Quando è necessario operare una modifica, il minor numero possibile di classi dovrebbero essere coinvolte

- La qualità del proprio codice si osserva proprio quando sorge l'esigenza

di fare modifiche