DE: Department of Engineering Ferrara

Ingegneria del Software Avanzata A.A. 2021/2022

Design Pattern

Damiano Azzolini
damiano.azzolini@unife.it

Laurea Magistrale in Ingegneria Informatica e dell'Automazione - Università di Ferrara

Design Pattern I

I *design pattern* sono delle soluzioni a problemi di programmazione che solitamente si incontrano durante lo sviluppo di un software con un linguaggio di programmazione object-oriented.

Sono indipendenti dal linguaggio di programmazione: forniscono delle soluzioni *generali* che devono poi essere adattate per il linguaggio che si utilizza.

Obiettivi:

- Riutilizzo di codice
- Utilizzo di soluzioni già testate per problemi comuni, senza bisogno di sviluppare nuovo codice
- Riduzione errori nel codice

Design Pattern II

Principi fondamentali solitamente adottati:

- Separazione delle responsabilità: ogni componente del programma (classe, metodo, ecc) dovrebbe occuparsi di una singola responsabilità
- Separazione del “cosa” dal “come”: dovrebbero essere rese disponibili solamente le informazioni necessarie per eseguire un certo servizio (“cosa”) e non l’effettiva implementazione (“come”)

Design Pattern III

Tre tipi di design pattern:

- Behavioral: descrivono come le classi e gli oggetti comunicano ed interagiscono
- Creational: descrivono come per poter creare classi ed oggetti con diversi livelli di astrazione e complessità, riutilizzando codice esistente
- Structural: descrivono come assemblare classi ed oggetti complessi in maniera flessibile partendo da oggetti più semplici

Ulteriori dettagli: `https://www.gofpatterns.com/design-patterns/module2/three-types-design-patterns.php`

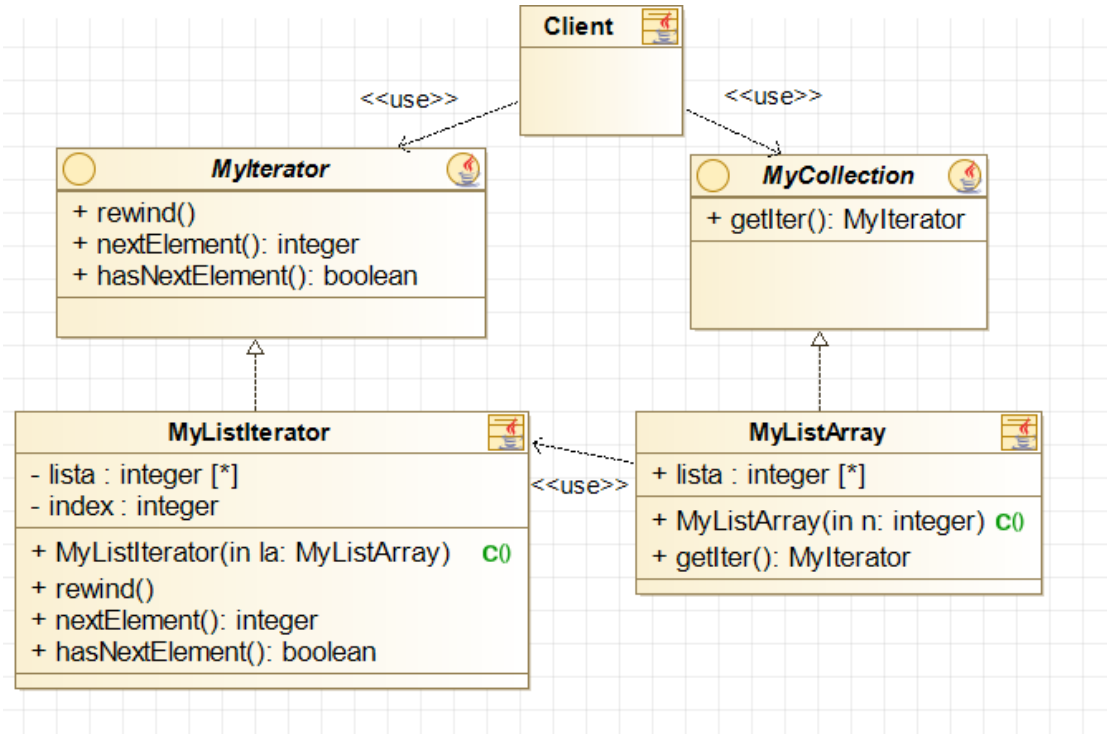
Design Pattern IV

Ci concentriamo su (ne esistono molti altri):

- Iterator pattern (behavioral): accesso elementi di una collezione in maniera sequenziale, senza conoscerne la rappresentazione sottostante
- Strategy pattern (behavioral): modifica del funzionamento di una classe o di un algoritmo a tempo di esecuzione
- Template pattern (behavioral): una classe astratta espone dei template per l'esecuzione di metodi che verranno sovrascritti o implementati dalle sottoclassi
- Decorator pattern (structural): aggiunta di funzionalità ad un oggetto senza alterarne la struttura

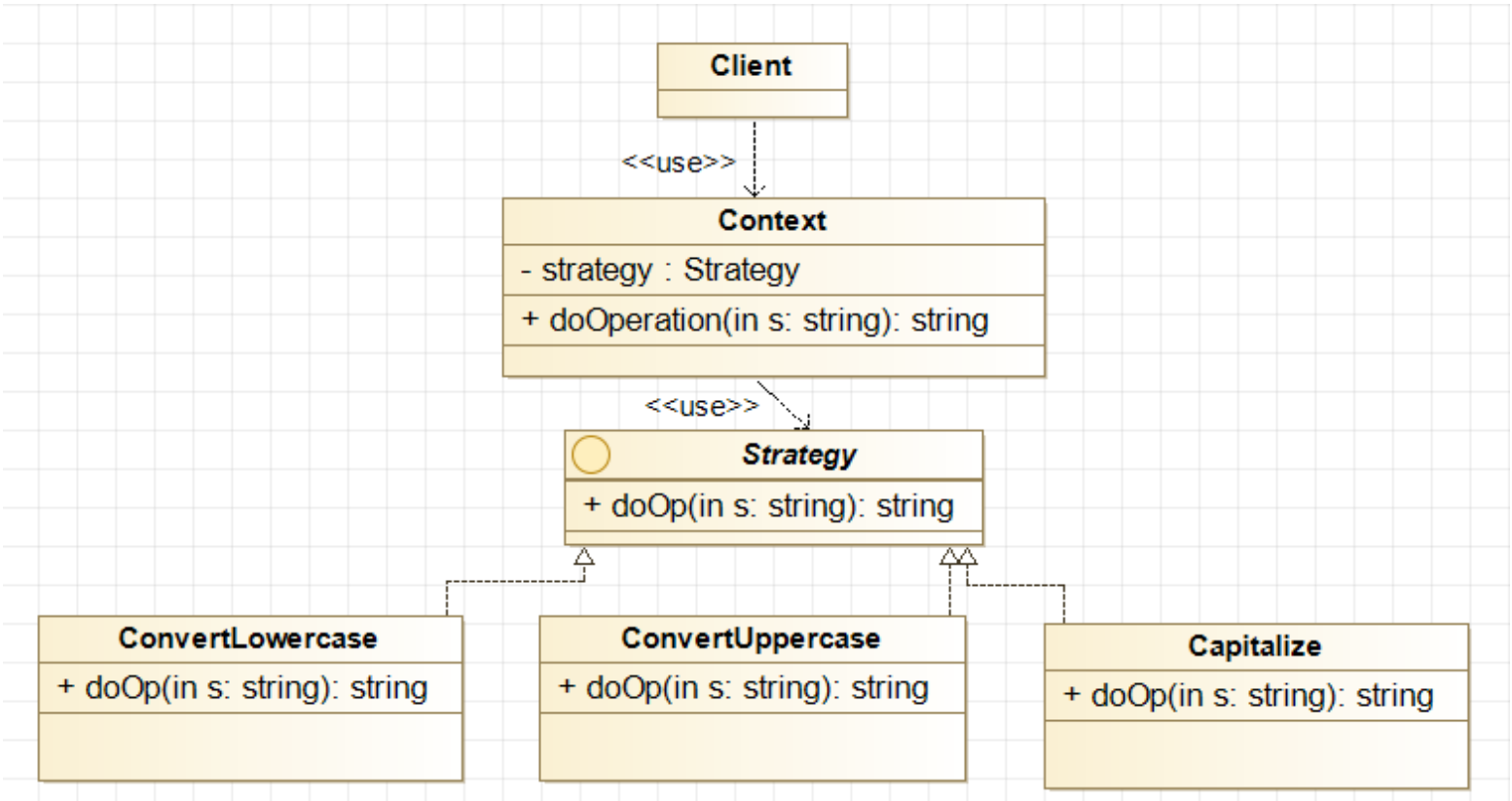
Iterator Pattern

Obiettivo: poter iterare su elementi di una collezione senza esporne l'implementazione. In altre parole, se cambia la struttura dati non cambiano i metodi.



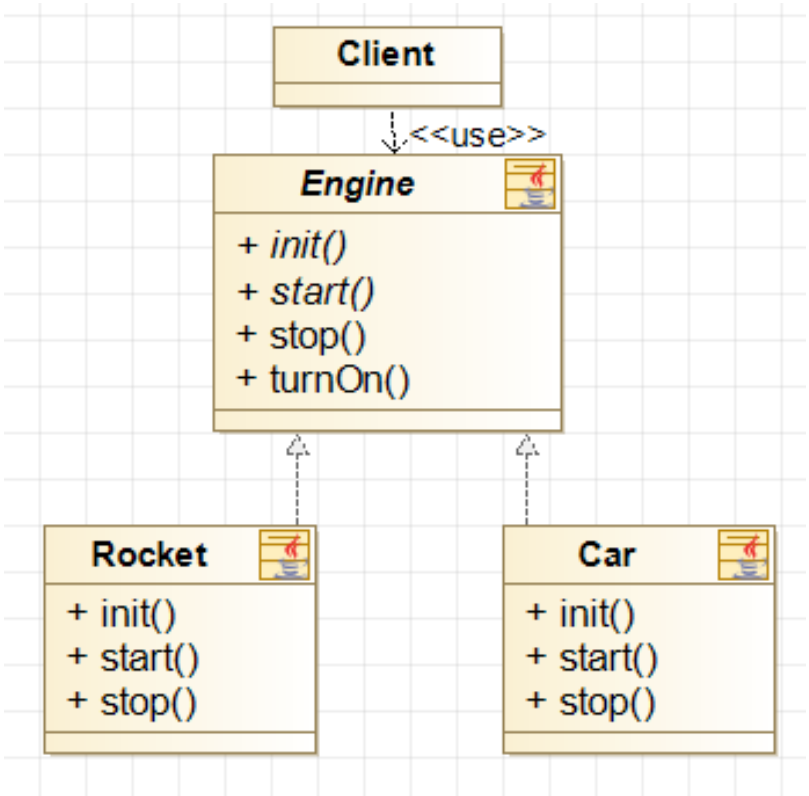
Strategy Pattern

Obiettivo: cambiare il comportamento di una classe o di un suo algoritmo a runtime.



Template Pattern

Una classe *astratta* espone delle classi e dei metodi. Il corpo principale del metodo viene definito `final` e l'implementazione di alcune parti viene demandata alle sottoclassi. Le sottoclassi possono sovrascrivere alcuni metodi, ma l'invocazione deve essere eseguita come definito nella classe astratta.



Decorator Pattern I

Descrive la possibilità di aggiungere comportamenti ad un oggetto senza alterare il comportamento degli altri oggetti della stessa classe.

Passi:

- Creazione di un'interfaccia I
- Creazione di classi concrete che implementano l'interfaccia I
- Creazione di una classe decorator D *astratta* che implementa l'interfaccia I
- Creazione di una classe concreta che implementa la classe astratta decorator D
- Utilizzo della classe astratta decorator per ampliare il funzionamento dell'interfaccia I

Decorator Pattern II