**Pattern**

“Un design pattern descrive soluzioni semplici e eleganti che descrivono problemi specifici in ambito OOP”

Dynamic binding risoluzione runtime dei tipi.

Una variabile di tipo di una classe base può essere usata per contenere una classe figlia.

L’implementazione di un sistema si dovrebbe focalizzarsi su poche classi base, e quelle specifiche dovrebbero essere usate in poche parti del codice, per favorire la manutenzione.

**Type check**

* Static check (Java): evita errori runtime, il compilatore verifica la validità della chiamata in fase di compilazione
* Dynamic check (CLOS, smalltalk): il controllo viene fatto runtime.

L’astrazione dei dati deve essere univoca per il programma.

Per favorire il riutilizzo delle astrazioni si usano i pattern, che riducono all’essenziale le astrazioni in modo generale.

Con i pattern si ha una documentazione ad alto livello, si ha maggiore flessibilità del codice, il codice è riutilizzabile….

Il pattern è una mini-architettura software.

I pattern facilitano la manutenzione del software, si fonde bene con la metodologia agile.

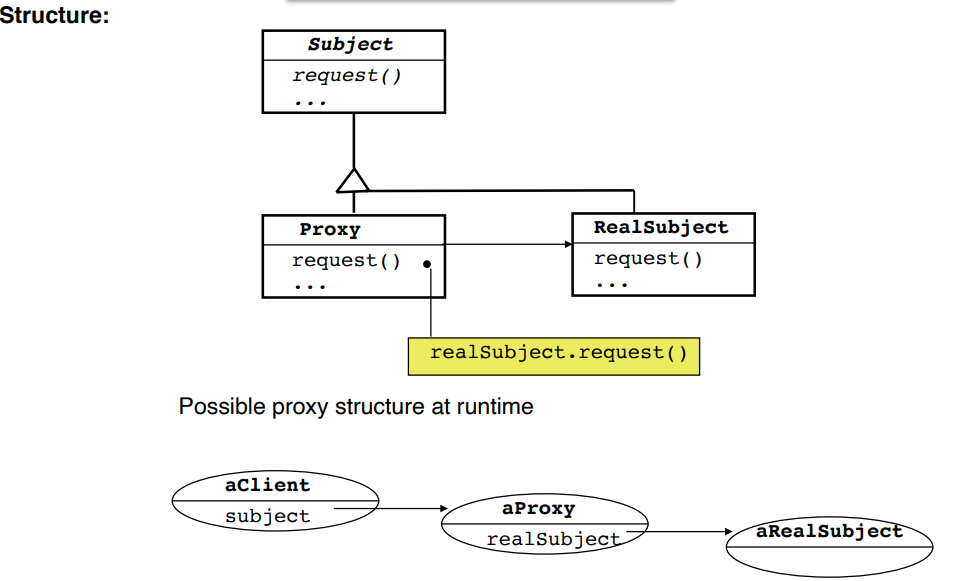
Elementi di un pattern

* Intenzione
* Nome alternativo
* Motivazioni che hanno portato alla creazione del patten
* Applicabilità
* Struttura delle classi
* Classi Partecipanti
* Collaborazione tra classi
* Conseguenze dell’utilizzo
* Tecniche di implementazioni
* Esempio di codice
* Casi d’uso noti
* Pattern simili

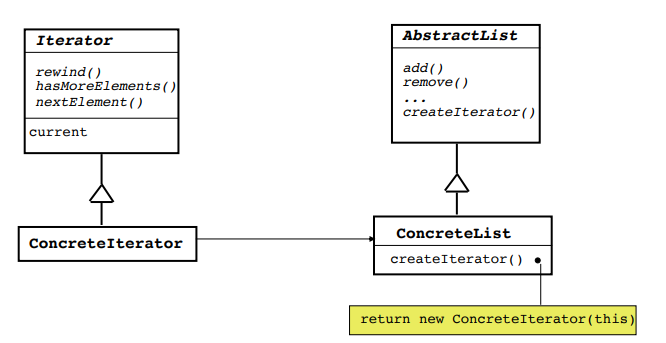
**Pattern Proxy**

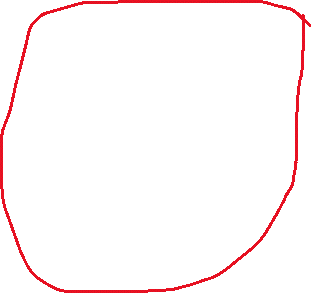
Fornisce un placeholder per un altro oggetto per controllarne degli altri.

Consente di gestire l’accesso ad un oggetto.



**Iterator pattern**

****



Se ogni tipo di lista ha un suo iteratore specifico, la creazione può essere delegato alla lista tramite un factory method.