

Laboratorio di Progettazione di Sistemi Software

Design Patterns

Valentina Presutti (A-L)
Riccardo Solmi (M-Z)

Indice degli argomenti

- **Tipi di Design Patterns**
 - Creazionali
 - Strutturali
 - Comportamentali

Alcuni concetti

▪ Incapsulare

- Racchiudere in modo da mostrare all'esterno solo quello che voglio. Mi servo di package, classi e modificatori di visibilità.

▪ Delegare

- Quando un metodo chiama un secondo metodo (di solito su un altro oggetto) per fargli fare il proprio lavoro.

▪ Inoltrare (forwards)

- Forma più debole di delega in cui la variabile *this* resta legata all'oggetto chiamato. NB In Java si può solo inoltrare una chiamata.

▪ Indirezione

- Ogni volta che prima di fare la cosa richiesta ne faccio un'altra (che in genere mi serve per capire come farla).
- Esempi: delega, inoltro e codice condizionale

Alcuni concetti /2

■ Variabilità: statica/dinamica

- *dinamica* significa che può cambiare durante l'esecuzione (a runtime) mentre
- *statica* significa che può variare solo prima dell'esecuzione (a compile time o a initialization time).

■ Con/senza stato (statefull/stateless)

- Classi (e quindi oggetti) con o senza campi.
- Cambia la riusabilità degli oggetti

■ Accoppiamento (coupling)

- Una classe/interfaccia è accoppiata ad un'altra quando contiene un suo riferimento. Esempio: chiamata con parametro di un tipo non primitivo.

Design Patterns

▪ **Definizione**

- Un Design Pattern descrive un problema ricorrente di progettazione e uno schema di soluzione per risolverlo.
- NB Il termine Design nel nome non significa che riguardano solo la progettazione

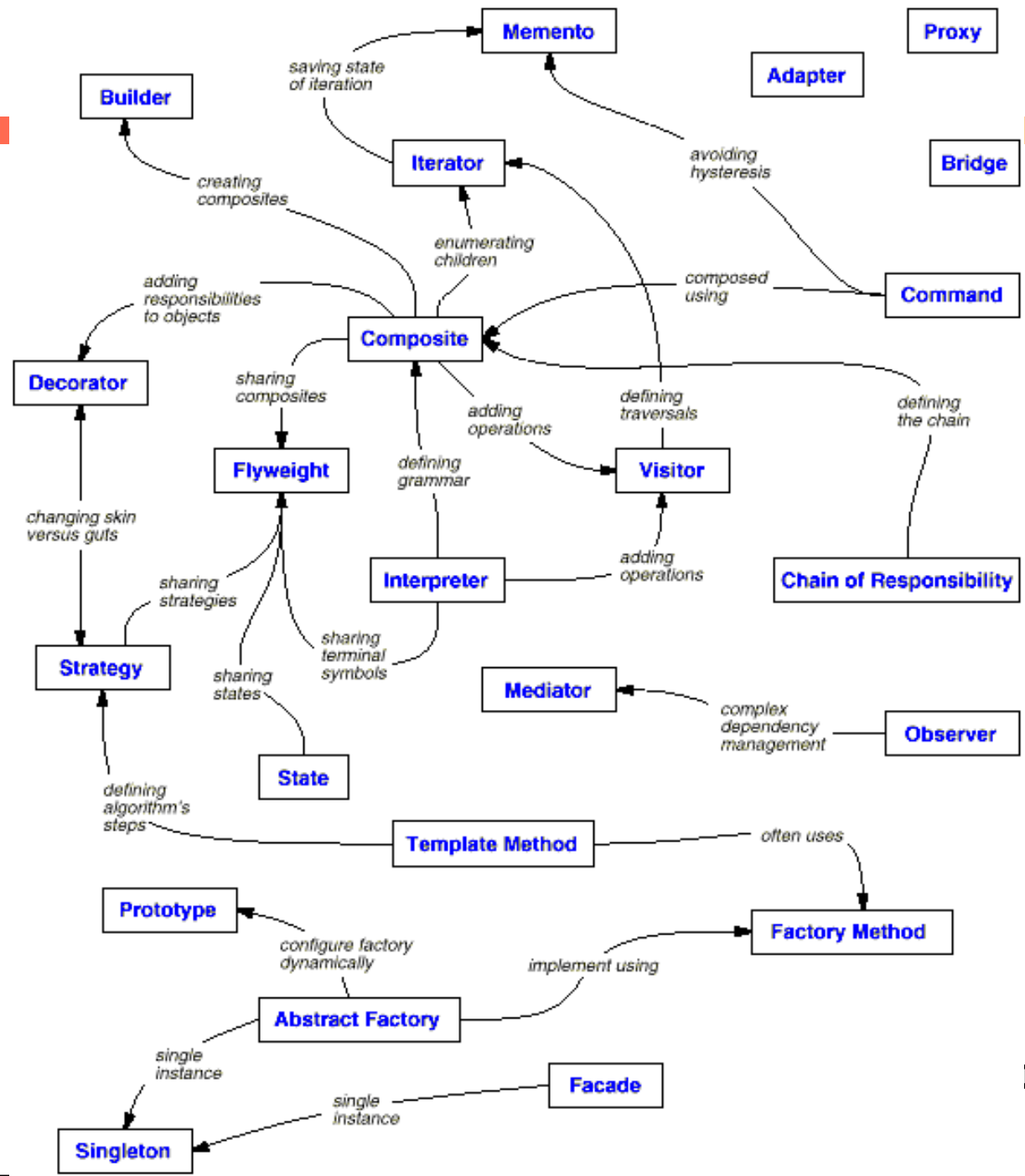
▪ **Serve a ...**

- Progettare e implementare il Software in modo che abbia la flessibilità e le capacità di evolvere desiderate
- Dominare la complessità di un Sistema Software

▪ **Catalogo**

- Ogni Design Pattern ha un *nome*, un campo di *applicabilità*, uno schema di *soluzione* e una descrizione delle conseguenze della sua applicazione.

Esempi di Design Patterns



Design Patterns

■ **Design Patterns: progettare anticipando i cambiamenti**

- *Prima* si progetta il Software poi lo si implementa.
- Massimizzare il riuso (senza ricompilazione)
- La compatibilità all'indietro è spesso un obiettivo.
- Assumo di poter controllare solo una parte del codice sorgente.
- Progettare il Sistema in modo da anticipare i nuovi requisiti e i cambiamenti a quelli esistenti.

Design Patterns

- **Usare Design Patterns per le esigenze attuali**
 - Riduco enfasi sulla progettazione iniziale
 - Mi basta progettare una soluzione ragionevole non la migliore soluzione.
 - Mi chiedo come evolverà il sistema ma mi basta sapere di poter aggiungere un certo tipo di flessibilità con il Refactoring
- **Incorporare Design nei sorgenti**
 - Le scelte progettuali e le dipendenze devono essere il più possibile espresse nei sorgenti
 - I commenti non sono un deodorante per rendere accettabile un codice illeggibile

Rischi e limiti dei Design Patterns

■ **Esistono dei trade-off**

- Non esiste una combinazione di Patterns che mi dà il massimo di flessibilità e di facilità di evolvere.
- Un Design Pattern facilita alcuni sviluppi futuri a scapito di altri.
- Cambiare una scelta progettuale può essere costoso

■ **Rischio di sovra-ingegnerizzare**

- Il codice diventa più flessibile e sofisticato di quanto serve.
- Quando scelgo i Patterns prima di iniziare ad implementare.
- Quando introduco i Patterns troppo presto.

■ **Costo computazionale**

- La flessibilità di una soluzione costa ed è accompagnata da una maggiore complessità.
- Usare un Design Pattern per esigenze *future* è un *costo*

Limiti OO evidenziati dai Design Patterns

▪ **Identità degli oggetti**

- Spesso una entità del dominio viene rappresentata da un insieme di classi dell'applicazione (object schizophrenia)
- Problema del significato di *this*.
- Problema della consistenza delle operazioni

▪ **Incapsulamento dei dati**

- La scomposizione di una entità in più classi mi obbliga a ridurre l'incapsulamento dei dati.

▪ **Rappresentazione indiretta**

- Dato un programma, non è facile stabilire quali Design Pattern sono stati applicati.
- Parte delle informazioni di progettazione sono perse.

Dipendenze nel codice

■ Scrivere un programma introduce dipendenze

- Comprendere le *dipendenze* che si introducono in modo da scegliere consapevolmente i Design Pattern.
- Le dipendenze contenute in una classe/interfaccia consistono nell'insieme dei riferimenti (statici) ad altre classi/interfacce.
- I design pattern vengono classificati in base al tipo di dipendenze su cui agiscono in: creazionali, strutturali, comportamentali.
- Esempi di dipendenze: istanziazione oggetti, campi e metodi, variabili e parametri di tipi non primitivi

Design Patterns Creazionali

- **Astraggono il processo di istanziazione**
 - Nascondo i costruttori e introduco dei metodi al loro posto
- **Conseguenze:**
 - Incapsulano la conoscenza delle classi concrete usate
 - Nascondono il modo in cui le istanze di queste classi vengono create e assemblate
 - Espongono interfacce al posto di tipi concreti
- **Esempi:**
 - Abstract Factory, Builder, Factory Method, Prototype, Singleton

Design Patterns Strutturali

- **Descrivono come comporre classi e oggetti in strutture più grandi**
 - Si dividono in: basati su classi e basati su oggetti
 - I primi usano l'ereditarietà (statica)
 - I secondi usano la composizione (anche dinamica)
- **Conseguenze**
 - Aggiungono un livello di indirizzione per disaccoppiare le due interfacce.
- **Esempi:**
 - Adapter, Composite, Decorator, Proxy

Design Patterns comportamentali

- **Riguardano algoritmi e l'assegnamento di responsabilità tra oggetti**
 - Descrivono pattern di comunicazione tra oggetti.
 - Per distribuire il comportamento usano l'ereditarietà o la composizione.
- **Conseguenze**
 - Alcuni incapsulano il comportamento in modo da poterlo variare anche dinamicamente
 - Disaccoppiano il chiamante e il chiamato aggiungendo un livello di indirectione.
- **Esempi:**
 - Command, Iterator, Observer, Template Method, Visitor