

Presentazione Pattern Abstract Factory

Andrea Bonisoli

Università degli Studi di Brescia,

Corso di Ingegneria del Software

30 marzo 2011



1 Introduzione

- Classificazione
- Scopo

2 Esempi

- Un cattivo esempio
- Un buon esempio
- Un secondo esempio

3 In generale

- I compiti
- La sua applicabilità

4 Conseguenze

- Vantaggi
- Limiti
- Pattern Correlati

5 Conclusioni

Introduzione

Classificazione

Abstract Factory

- È un pattern di creazione, ovvero ci aiuta per generare istanze di altre classi.
- È basato sugli oggetti, ovvero la generazione di istanze è delegata ad appositi oggetti.
- Viene anche chiamato *Kit*.

Come si può intuire dal nome utilizza classi astratte (o meglio ancora potrebbe utilizzare interfacce) le quali verranno poi implementate da altrettante classi concrete. Sfrutta poi la delega dai metodi astratti alle implementazioni concrete e per questo motivo è basato sugli oggetti.

Introduzione

Scopo

Secondo la Gang of Four questo pattern dovrebbe

*“Fornire un’**interfaccia per la creazione di famiglie di oggetti correlati o dipendenti** senza specificare quali siano le loro classi concrete”.*

Inoltre può essere usato per di schermare i client da differenti piattaforme che forniscono differenti implementazioni per uno stesso insieme di concetti.

Vediamolo meglio con alcuni esempi.

Esempi

Un cattivo esempio

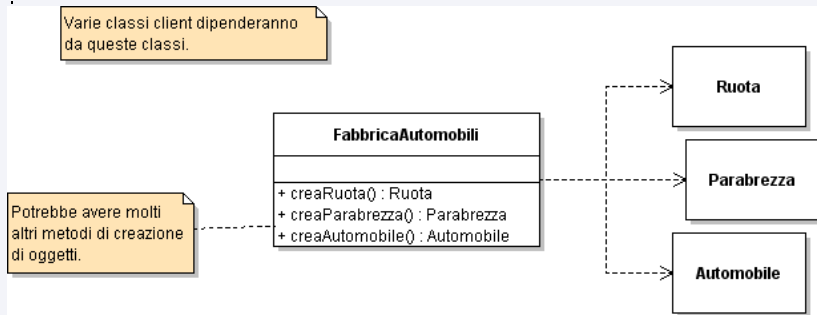
Fabbrica

Sia dal pattern Factory Method, sia da casi reali, viene suggerito l'utilizzo di una *fabbrica* che si occupi di istanziare le varie classi: questa fabbrica consente di mantenere meglio il controllo sulle varie istanze ed è l'unica autorizzata a crearle.

Fabbrica di automobili

Supponiamo di avere un software contenente una classe che rappresenta una fabbrica di automobili: questa fornisce non solo le automobili complete ma anche i vari pezzi di ricambio. Tutte le classi che richiedono un pezzo oppure un'automobile saranno quindi i client che accedono ai suoi metodi, ognuno dei quali si occupa di produrre uno specifico pezzo.

Una possibile implementazione potrebbe essere rappresentata dal seguente diagramma delle classi:



Non si tratta naturalmente del diagramma completo.

Problemi

Un siffatto progetto tuttavia possiede un certo numero di difetti. In particolare risulta difficile da modificare per estenderlo con nuove tipologie di automobili. Infatti se volessimo poter produrre una berlina? Oppure una station wagon? Oppure delle ruote cromate? Oppure...

Cattive soluzioni

Si potrebbe pensare di creare nuovi metodi appositi nella classe `FabbricaAutomobili`.

Oppure di parametrizzare ogni metodo perché possa sapere quale tipo di automobile produrre.

Ma tutto questo non produce del buon codice!!

Dov'è il vero problema?

Questi problemi non derivano dall'utilizzo di una fabbrica, ma dalla sua implementazione troppo *inflexibile*.

Infatti così implementata la fabbrica deve necessariamente conoscere **tutte** le tipologie di automobili, di ruote, di parabrezza, eccetera...

Per risolvere questo problema sarebbe meglio rendere la fabbrica indipendente dai dettagli degli oggetti creati, spostando la logica di creazione presso un'altra classe.

Sfruttando i vantaggi della programmazione orientata agli oggetti vogliamo quindi **delegare** la creazione, mantenendo solamente l'interfaccia della fabbrica e degli oggetti creati (che chiameremo prodotti).

Questa soluzione si chiama: **Abstract Factory!**

Esempi

Un buon esempio

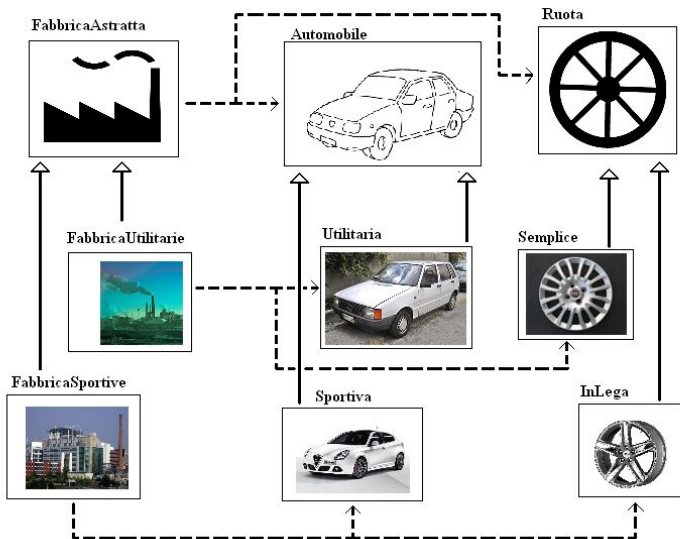
Fabbrica astratta e fabbriche concrete

Rivediamo quindi la nostra fabbrica: questa verrà sostituita da una fabbrica astratta, così come i suoi prodotti saranno solamente astratti (o eventualmente delle interfacce).

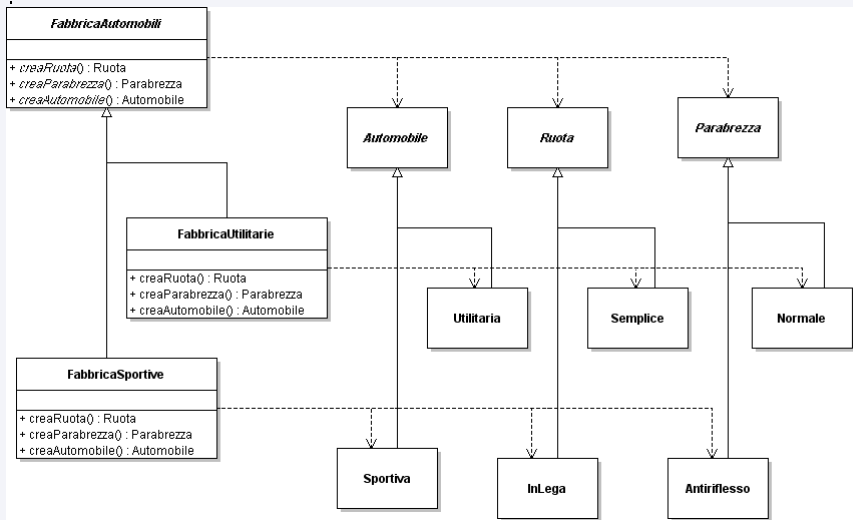
Ad essa affiancheremo delle fabbriche concrete che producono prodotti concreti, implementazioni delle relative classi astratte.

Fabbriche diverse potranno produrre prodotti diversi, ma tra loro simili (infatti avranno la stessa interfaccia). Ogni fabbrica si farà carico di una tipologia di prodotto e sarà indipendente dalle altre, così come pure il client sarà indipendente dalle varie implementazioni: dovrà solo scegliere a quale fabbrica concreta rivolgersi. E se dovesse “cambiare idea” gli basterebbe rivolgersi ad una diversa fabbrica concreta.

Vediamo questa idea tramite delle immagini:



O più precisamente con il seguente diagramma delle classi:



Esempi

Secondo esempio

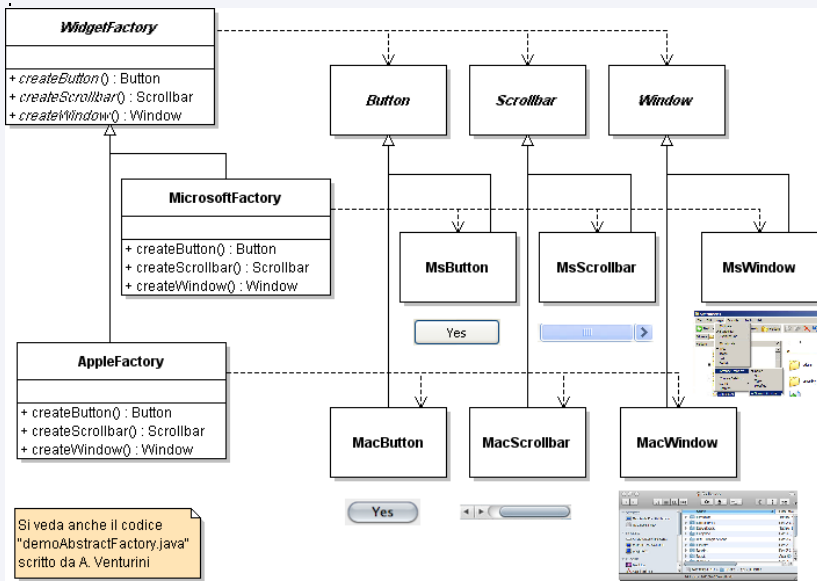
Widget Factory

Una seconda tipica applicazione di questo pattern sono le *Widget Factory* indipendenti dalla piattaforma.

Infatti è spesso utile una fabbrica di widget tra loro consistenti che sia indipendente dall'implementazione o che possa funzionare su diverse piattaforme.

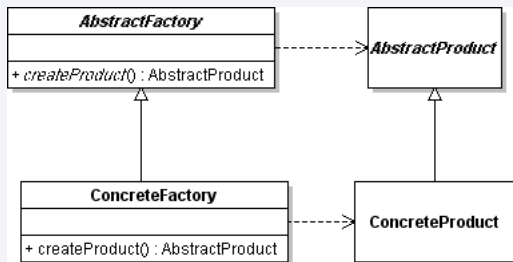
Come possiamo quindi rendere una applicazione che necessita di bottoni, scrollbar, menù e finestre indipendente dal sistema desktop utilizzato, consentendole di ignorare quale particolare famiglia di widget stia utilizzando e di non occuparsi delle operazioni per creare questi elementi?

Ecco un possibile diagramma delle classi:



Il diagramma generale

Quindi in generale avremo una fabbrica astratta con dei metodi per creare oggetti astratti ed una o più fabbriche concrete, implementazioni di quella astratta, che produrranno oggetti concreti, implementazione dei relativi astratti.



I compiti

Le fabbriche ed i prodotti

- **AbstractFactory** dichiara un'interfaccia per la creazione di oggetti prodotto astratti
- **ConcreteFactory** implementa le operazioni di creazione degli oggetti prodotto concreti
- **AbstractProduct** dichiara un'interfaccia per una tipologia di oggetti prodotti
- **ConcreteProduct** definisce un oggetto prodotto che verrà creato dalla corrispondente factory concreta

I client

Tutte le classi che richiedono la creazione di oggetti conosce ed utilizza soltanto le interfacce dichiarate dalle classi AbstractFactory e AbstractProduct

Quando è bene applicare questo pattern?

- 1 Quando vogliamo un sistema **indipendente** dalla modalità di creazione dei suoi „prodotti”
- 2 Quando vogliamo avere la possibilità di scegliere tra più famiglie o „kit” di prodotti
- 3 Quando vogliamo essere sicuri di usare solo i prodotti della famiglia scelta
- 4 Quando abbiamo una libreria di classi di cui vogliamo rivelare solo le interfacce, non le implementazioni

Conseguenze

Vantaggi

Anzitutto **isola** le classi concrete dei prodotti garantendo indipendenza tra i client e le classi effettivamente utilizzate per l'implementazione dei prodotti.

Infatti un client manipola le istanze dei prodotti solo attraverso le loro interfacce astratte mentre la creazione di nuovi prodotti è responsabilità delle fabbriche.

Inoltre consente di **cambiare** in modo semplice la famiglia di prodotti utilizzata: generalmente in una applicazione solamente in un punto compare la scelta della fabbrica concreta e spesso questa scelta può anche essere fatta runtime in base alla configurazione.

Infine promuove la **coerenza** nell'utilizzo dei prodotti e tra i prodotti stessi:

- i prodotti di una stessa fabbrica concreta appartengono ad una stessa famiglia e sono progettati per essere utilizzati insieme (ad esempio i widget di un kit hanno un *look and feel* coerente);
- quando un'applicazione sceglie una fabbrica concreta, utilizzerà solamente i prodotti di quella fabbrica e non dovrà e neppure potrà mescolare prodotti differenti.

Ma questo ultimo punto potrebbe anche essere uno svantaggio?

Conseguenze

Limiti

Effettivamente questo pattern ha alcuni limiti. Essi sono causati sostanzialmente dalla **difficoltà di aggiungere** nuovi prodotti. Poiché il set di prodotti è determinato dall'interfaccia della fabbrica astratta, per aggiungerne uno dobbiamo modificare questa interfaccia e, di conseguenza, tutte le fabbriche concrete!

Allo stesso modo è sconsigliabile disporre di una fabbrica mista, ovvero che produce oggetti appartenenti a diversi kit dato che questo violerebbe l'indipendenza tra le diverse fabbriche concrete e tra i prodotti concreti di diverse famiglie.

Fortunatamente questi limiti si possono aggirare, se necessario, attraverso l'uso di **fabbriche estendibili** (poco consigliabile) o combinando questo pattern con il pattern **Prototype**.

Conseguenze

Pattern Correlati

Prototype

Questi due pattern possono essere combinati realizzando una sola fabbrica concreta che sia anche parametrica: essa conterrà delle istanze prototipo (un'istanza prototipo per ogni tipo di prodotto) che le sono state fornite al momento della sua inizializzazione. Istanze prototipo che le sono fornite quando verrà a sua volta inizializzata. L'oggetto fabbrica concreta utilizzerà tali prototipi per creare nuovi prodotti, *clonando* il prototipo apposito per costruire la nuova istanza.

Singleton

Normalmente, ad una applicazione serve una sola istanza di una fabbrica concreta: combinando quindi questi due pattern possiamo assicurarci che esista al più una sola istanza di ogni fabbrica concreta.

Conseguenze

Pattern Correlati

Façade

Questo pattern definisce un'interfaccia unificata attraverso cui accedere all'intero sotto-sistema. I client comunicano con questa interfaccia la quale delega una richiesta ad altri oggetti. Allora il pattern Abstract Factory può essere utilizzato come una variante creazionale del pattern Façade.

Factory Method

Questo pattern ha scopi molto simili e spesso viene utilizzato durante l'implementazione delle singole fabbriche concrete. Si noti tuttavia che Factory Method è un pattern basato su classi, non sugli oggetti.

Conclusioni

Riassumendo

Riassumendo i concetti chiave

- ❶ **Delega della creazione:** la responsabilità della creazione di nuovi oggetti è incapsulata in classi apposite
- ❷ **Famiglie di oggetti:** ci si assicura che i prodotti usati appartengano alla stessa famiglia
- ❸ **Indipendenza:** i client dipendono solamente dalle interfacce, non dalle implementazioni

Lo scopo dichiarato era

“Fornire un'interfaccia per la creazione di famiglie di oggetti correlati o dipendenti senza specificare quali siano le loro classi concrete”.