Idiomatica: codice orribile, antipattern e pratiche discutibili

In questa rubrica analizzeremo i più comuni errori di codifica, di progettazione ed i tranelli in cui è facile cadere. Programmatore avvisato, mezzo salvato!

La massima di questo mese:

> “A worker may be the hammer's master, but the hammer still prevails. A tool knows exactly how it is meant to be handled, while the user of the tool can only have an approximate idea”
> *M.Kundera*

#3: Falsi Oggetti: oggetti inutili, anemici, tristi. E pericolosi.

I Falsi oggetti sono molto comuni se non si applicano correttamente i principi di buon design Object Oriented. I falsi oggetti sono spesso associati con il polimorfismo dei poveri, che ho trattato nel primo numero di questa rubrica, e qualche volta ne sono addirittura dei generatori.

Ma ripassiamo un momento uno dei cardini della programmazione Object Oriented: un oggetto è fatto di comportamento, o *behaviour*. Questo comportamento tipicamente si applicherà su dei dati, che altrettanto tipicamente saranno contenuti nell'oggetto stesso. Da qui il vecchio adagio che un oggetto è composto da dati + comportamento, adagio tanto noto quanto poco applicato nella pratica.

Se volessimo riassumere in una frase sola cos'è che caratterizza un buon design ad oggetti, potremmo dire – magari semplificando un po' - che un buon design è caratterizzato dal poter aggiungere una feature ad un software semplicemente aggiungendo un nuovo oggetto, senza toccare tutto il resto. E possibilmente senza dover ricompilare l'esistente.

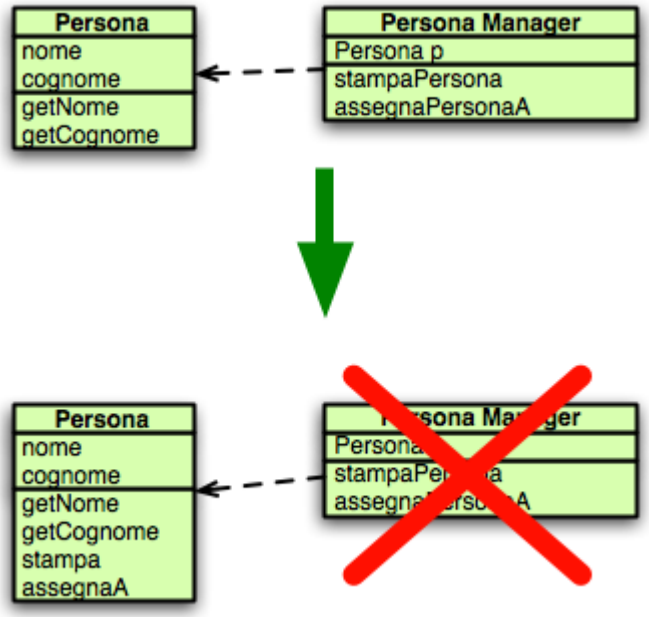
Dovrebbe quindi essere evidente che il polimorfismo è essenziale per raggiungere questo scopo, addirittura imprenscindibile. Da qualche parte nel nostro codice ci sarà infatti un oggetto che, tramite un'astrazione, dovrà essere capace di applicare delle logiche che ancora non sono state scritte ed implementate: in un certo senso, sarà a prova di futuro. In un linguaggio fortemente tipato come Java, senza interfacce e polimorfismo sarebbe impossibile scrivere codice estensibile. Ma, continuando nel ragionamento, il polimorfismo implica ovviamente un comportamento da rendere polimorfo, e non dei semplici metodi *accessors* (o getter/setter che dir si voglia). Lungi da me affermare che gli accessors siano sempre un errore di design: hanno i loro utilizzi. Quel che è certo è invece che un oggetto che non ha un comportamento, un oggetto che non *fa* nulla, è di solito un segno di programmazione strutturata.

Per cercare nel codice questo problema, seguire i passi:

1. L'Oggetto x della classe X non ha comportamento, solo getters/setters.
2. Una logica, un comportamento da qualche parte ci sarà pure, oppure il software non avrebbe senso
3. Questo comportamento sarà sparso in altre classi, che probabilmente avranno dei nomi simili a XHandler, XManager, ecc.
4. Nel caso peggiore, da qualche parte ci sarà qualcuno che fa un instanceof, ricadendo addirittura nel Polimorfismo dei poveri (vedi Idiomatica #1)

A parte i (pochi) casi in cui un oggetto anemico è giustificato, la soluzione è decisamente semplice. Il comportamento, sparso qui e la, va spostato nel codice delle classi in cui è più appropriato, tipicamente quelle che hanno i dati di cui il comportamento stesso ha bisogno. O, in alternativa, la classe va fatta sparire del tutto. Dunque

5. Se non ci sono validi motivi per non farlo, spostare il comportamento dalle classi Handler/Manager alla classe di dominio



Esercizi:

- Andare a cercare i falsi oggetti nel vostro codice
- Andare a cercare i falsi oggetti in un software open source a piacere
- Elencare i casi in un oggetto anemico è giustificato dal contesto

Conclusioni:

I Falsi oggetti sono molto comuni se non si applicano correttamente i principi di design. Stranamente, sono anche molto comuni fra chi *pensa* di applicare dei principi di design: tipicamente, succede quando si da troppa importanza agli strati di Service, magari fuorviati da qualche acronimo di moda di tre lettere