Le interfacce nella OOP ad ereditarietà singola

Cosa è un interfaccia?

Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

from Sun Java Tutorial..

Un interfaccia permette ad una classe di diventare più formale circa il comportamento che deve fornire.

L’interfaccia è una forma di contratto tra la classe ed il mondo esterno e questo contratto viene rispettato in fase di costruzione della classe da parte del compilatore. Se la classe usa un’interfaccia, tutti i metodi definiti nell’interfaccia devono apparire implementati nel codice sorgente della classe prima di procedere alla compilazione.

Perché è stato utilizzato il meccanismo delle interfacce nei linguaggi di programmazione che implementano l’ereditarietà singola?

La prima vera motivazione è sopperire ai limiti dell’ereditarietà singola che come sappiamo non permette ad una classe figlio di ereditare più classi padre.

Questa peculiarità è esclusiva dell’ereditarietà multipla.

La seconda motivazione è la gestione dei casi particolari che non possono essere inseriti in una gerarchia di classi.

Questi casi possono verificarsi anche per classi di tipo diverso ed essere quindi comuni a classi completamente diverse.

La terza motivazione è l’impossibilità di conoscere il codice che un metodo dovrà implementare e volete che tale metodo venga ereditato da una classe.

Facciamo degli esempi!

Nella circolazione delle persone tra i vari paesi del mondo, è necessario per molti di essi un visto, ma può capitare che le persone di una determinata nazione siano esenti dal rilascio del visto.

Una persona dell'Unione Europea o degli USA può andare in Germania senza visto, mentre una persona della Confederazione Russa ha bisogno di visto.

Nella rappresentazione della gerarchia delle classi visto potrebbe essere rappresentato come intefaccia chiamata IVisto.

Un interfaccia nel Framwork .NET ha come convenzione l’uso delle lettera I davanti al nome dell’interfaccia.

E se voleste scrivere del codice per gestire il rilascio e la validità dei visti fornendolo poi ad una classe figlia, come fareste?

Per il lettore che non conosce questo aspetto ricordo che esistono una moltitudine di tipi di visti, da quello turistico a quello sportivo, da quello medico a quello per lavoro autonomo ed ognuno di questi sono vincolati da un periodo di validità del visto stesso che comporta il periodo che lo straniero può stare nel paese in cui si reca. Un italiano può andare in ferie a Mosca per un periodo massimo di 90 giorni con apposito visto.

Essendo visto quindi un caso non generale per tutte le persone ed essendo successivamente impossibile determinare a priori il tipo e la durata del visto che verrà rilasciato, l'ipotesi di candidare "Visto" ad interfaccia è la scelta più logica e sensata.

Da questo esempio capite che:

1. Visto è un caso particolare
2. Non sapendo il tipo e la durata del visto è impossibile scrivere subito il codice per la gestione di due ipotetici metodi “get_tipovisto()” e “get_durata()”.
3. Dichiarando “IVisto” come interfaccia stipulate un contratto con la classe che la eredita la quale sarà obbligata ad implementare i due metodi sopra citati.
4. Abbiamo sopperito alla mancanza dell’eredità multipla che poteva fornirmi le informazioni inerenti il visto.

Facciamo attenzione al punto numero 4 dell'elenco precedente perché se visto è un caso particolare, andare ipoteticamente ad inserire in una struttura gerarchica di classi ad ereditarietà multipla tali informazioni potrebbe rendermi, se mal progettata,ingestibile o incoerente l'intera struttura di classi.

L'ereditarietà multipla richiede un linguaggio che sia capace di gestirla correttamente in tutte le sue sfaccettature.

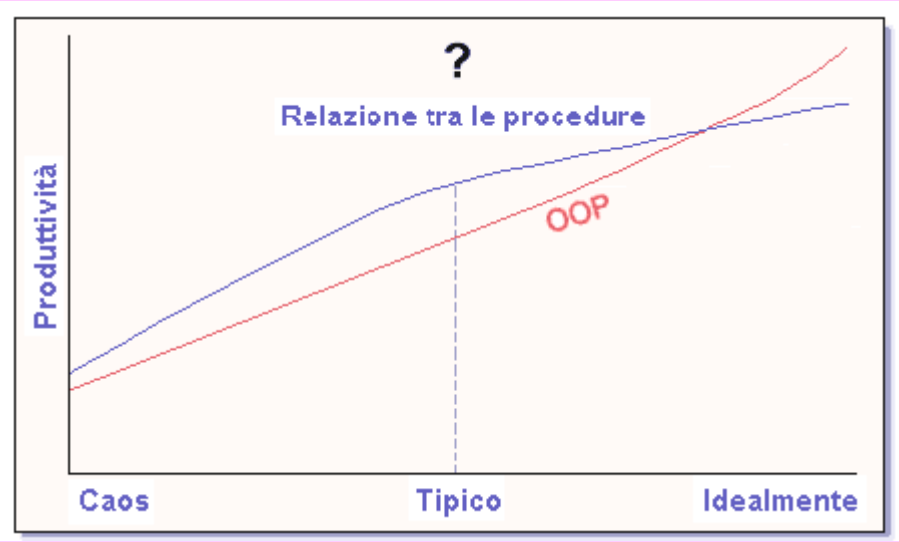
Ma siamo realmente obbligati ad usare la OOP ed il meccanismo di ereditarietà?

OOP vs. Prog. Procedurale

La risposta è NO!

L’analista deve valutare il problema che deve risolvere, perché la OOP non è la risposta a tutti i mali ed in molti casi non offre nessun beneficio, anzi rende lo sviluppo più complesso e meno flessibile.

Il culto della OOP



…ma sarà realmente così ?