Esercizi Programmazione

October 9, 2017

Impostazione

Il programma e' strutturato in modo da lanciare automaticamente i programmi gia' implementati e testarli con qualche input che ho scelto.
Usando il comando *make* potrai lanciare il programma che vuoi testare. Ad esempio

```
make 3
```

compilera' il file *ex03.java* e cerchera' di eseguirlo con vari test da me scelti. Considerero' un programma completo se ha passato tutti i test

Descrizione del problema

Troverai tutti i file nella directory *tasks* con gia' la classe principale implementata e la descrizione del problema.
Questa avra' un titolo e la consegna. l'ultima riga rappresentera' l'input che dovresti aspettarti con le seguenti convenzioni

- **int** per input intero
- **long** per input intero in doppia lunghezza
- **double** per input reale
- **string** per input simbolico

Se non e' presente nessun parametro, il programma non deve leggere alcun parametro

Soluzioni

Nel caso non riuscissi a risolvere il problema, ho fornito una mia soluzione nella directory *solutions*. Usala come ultima risorsa. Usa allo stesso modo il makefile e osserva le soluzioni da me fornite. Anche se il programma termina con successo, alcuni risultati potrebbero differire dai tuoi. Fai attenzione e in bocca al lupo!

Esercizi Java

Di seguito elechero' una serie di esercizi riguardante la Programmazione a oggetti (OOP). Questi richiedono una maggiore conoscenza di Java e possono essere in realta' essere risolti in modi diversi, non usando comunque la OOP, percio' cerca di usare il paradigma ad oggetti ove possibile

Ereditarieta'

```
Object
    AbstractCollection
        AbstractList
            ArrayList
```

Un elemento chiave della OOP e' la capacita' di creare superclassi estese di volta in volta in base al problema da risolvere. Cio' porta all'efficiente propagazione di una modifica verso tutte le sottoclassi: se in una nuova versione di Java si decide che un oggetto debba avere un parametro aggiuntivo, modificando la classe *Object* si propaga la modifica a tutto cio' che e' un oggetto (in questo caso veramente qualsiasi cosa). Cosi' come i miglioramenti si propagano anche gli errori, quindi parti disegnando bene il tuo sistema.
Cerchiamo di esplorare l'efficienza dell'ereditarieta'.

Polimorfismo

Attraverso le interfacce, e' possibile scrivere una serie di metodi che una classe deve avere. Una classe puo' implementare piu' interfacce ed e' quindi meno vincolante dell'ereditarieta' che la limita a una. Vedila piu' semplicemente come un pezzo di un puzzle che posso attaccare alla classe *"Se mi implementi questi metodi, ti posso considerare come quell'interfaccia"*
Inoltre una stessa interfaccia puo' essere implementata da piu' classi. Tutte queste classi percio' possono essere considerate come quella stessa interfaccia, ma sono effettivamente diverse. Un esempio sara' piu' chiaro:

```
Shape
 -> Circle
 -> Square
 -> Triangle
```

Tutte le forme, Circle, Square e Triangle sono classi diverse che implementano la stessa interfaccia *Shape*. E' possibile comunque avere un array di Shape e popolarlo con tutte e 3 le classi.

```
Shape s = new Triangle();
```

e' una sintassi perfettamente valida, poiche' Triangle puo' essere effettivamente visto come Shape.

Incapsulamento

Sempre attraverso l'ereditarieta', una sottoclasse non reinventa la ruota da 0 ma si ritrova a sua disposizione metodi e parametri che le superclassi hanno scelto di rendere pubblici o protetti. In questo senso tutti i dettagli implementativi delle superclassi vengono nascosti e ne viene quindi garantito il corretto funzionamento.

Esercizio 1

Implementare una classe *Person* strutturata come segue

```
class Person
    Person(String name, String surname)
    String getName()
    String getSurname()
    void sayHi()
```

dove sayHi() stampi a schermo il nome e il cognome della persona.

In seguito scrivi una classe *Student* che includa anche un campo intero *badge_number* con relativo getter (il metodo che restituisce il campo).
Sempre ereditando da *Person* crea anche una classe *Teacher* con un array di stringhe dei corsi di cui di occupa.
Nel main crea un array di *Person* e popolalo con 2 persone, 3 studenti e un insegnante. Tutti dovranno dire il proprio nome.
Ipotizza che passi del tempo e l'idea di persona non ti soddisfa piu': ti serve anche un campo che ne registri il codice fiscale. Effettua le dovute modifiche in modo che anche le sottoclassi siano preparate a ricevere un codice fiscale in input (crea altri costruttori o modifica l'originale).

- Nota che in tutte le tue istanze e' presente il metodo *sayHi()* grazie all'ereditarieta'. Nota anche che dopo aver fatto una modifica a un costruttore, sei costretto a rimaneggiare tutte le successive chiamate nella tua code base. Per piccoli progetti non e' un problema, ma quando lavori con decine di migliaia di righe di codice, preferiresti spendere il pomeriggio a lavorare a una nuova feature piuttosto che ripassare tutte le classi del progetto in cerca delle chiamate ai costruttori ;)