

Sistemi per l'Elaborazione dell'Informazione 2

Riflessività e oggetti per procura

Lo scopo di questa esercitazione è abituarsi all'uso di metodi per la gestione della riflessività.

Indovina la classe!

Lo scopo di questo programma è di realizzare un piccolo e divertente (?) gioco a due giocatori: il programma chiede il nome completo (cioè incluso il pacchetto) di una classe Java al primo giocatore, lo nasconde al secondo (potete andare a capo una quarantina di volte) e quindi mostra in bella forma al secondo giocatore la superclasse della classe data, tutte le interfacce implementate dalla classe e i metodi pubblici. A questo punto il secondo giocatore può cercare di indovinare (diciamo al più tre volte) la classe.

Fate attenzione alla differenza tra i metodi che restituiscono membri (cioè attributi, metodi o costruttori) dichiarati e metodi implementati: dovete usare i secondi (oppure non visualizzerete i metodi ereditati).

Alfabeti farfallini e non

Dovete scrivere un sistema di “farfallinamento” di un testo arbitrario. Non vogliamo però legarci al solito alfabeto farfallino (nel caso non ve lo ricordaste dal corso di C, l'alfabeto farfallino consiste nel sostituire a ciascuna vocale una sequenza di tre lettere della forma vocale-f-vocale. Per esempio, alla lettera *a* viene sostituita la sequenza *afa*, alla lettera *e* la sequenza *efe* e così via.

Vogliamo quindi specificare un'interfaccia **Alphabet** con un solo metodo **transform()** che accetta come argomento un carattere e restituisce una stringa (la trasformata del carattere). Dovete scrivere almeno due implementazioni di **Alphabet**, per esempio quelle farfallina e quella (notoriamente riservata agli imbranati) che usa la “p” al posto della “f”. Potete anche implementare altre trasformazioni (per esempio, alla “Garibaldi fu ferito”).

Ogni implementazione deve essere realizzata tramite un singoletto memorizzato in un campo statico finale **INSTANCE**, reperibile tramite un metodo di fabbricazione statico **getInstance()**.

A questo punto dovete fornire un metodo statico in una classe a parte, che chiameremo **AlphabetUtils**, che accetta come argomenti una stringa, il nome di un'implementazione di **Alphabet** (sempre sotto forma di stringa) e restituisce la stringa trasformata. Per fare questo, dovrete recuperare la classe tramite il metodo **forName()** e invocare il suo metodo statico **getInstance()**: notate che per invocare un metodo statico nel campo riservato all'oggetto dovete mettere l'oggetto di tipo **Class** che rappresenta la classe.

A questo punto basta scandire la stringa, ottenerne i caratteri tramite **charAt()** e concatenare in uno **StringBuilder** (che cos'è?) le stringhe ottenute applicando la trasformazione rappresentata dall'oggetto che avete recuperato per riflessività.

Infine, **AlphabetUtils** deve avere un metodo **main()** che legge dalla riga di comando (**args[0]**) il nome di una classe che implementa **Alphabet**, trasforma tutti i parametri rimanenti (da **args[1]** fino ad **args[args.length-1]**) e li stampa.

Varianti

Potete omettere il metodo di fabbricazione e rendere pubblico il campo `INSTANCE`. In questo caso, per recuperare l'oggetto che vi serve dovrete usare `getField()` e i metodi di `Field`.

Proxy

Il package `java.lang.reflect` contiene, tra l'altro, la classe `Proxy` e l'interfaccia `InvocationHandler`, le quali permettono di implementare oggetti per procura. In particolare, il metodo di fabbricazione `newProxyInstance()` crea oggetti `Proxy` una volta che sono stati specificati:

1. un *class loader* dell'oggetto di cui si vuole ottenere una procura (che risulta facilmente ottenibile invocando il metodo `getClassLoader()` dell'oggetto `Class` corrispondente;
2. un array di interfacce implementate dall'oggetto per cui si vuole costruire una procura;
3. un oggetto di una classe che implementa `InvocationHandler`, il cui metodo `invoke()` specifica come agirà la procura. Ogni volta che viene eseguito uno dei metodi dell'oggetto originale, se questo appartiene alle interfacce selezionate quando si è creato l'oggetto `Proxy` verrà automaticamente eseguito `invoke()`, passandogli:
 - (a) un riferimento all'oggetto `Proxy` che si sta utilizzando;
 - (b) un riferimento all'oggetto `Method` corrispondente al metodo che è stato invocato;
 - (c) un array di istanze di `Object` contenente gli argomenti specificati.

L'esercizio consiste nel generare un'istanza di `Proxy` in grado di stampare i metodi invocati su un oggetto *e* il valore da loro restituito.

Definite una classe `Debug` che contiene un solo metodo statico di fabbricazione `makeDebuggable()`. Il metodo accetta un oggetto e restituisce un oggetto, generato da `Proxy`, *che implementa tutte le interfacce dell'oggetto originale*. Per ottenere questo risultato, dovete rilevare le interfacce implementate tramite riflessività (utilizzando `getInterfaces()`).

Chiaramente, l'`InvocationHandler` passato al costruttore di `Proxy` deve tenere traccia dell'oggetto originale, invocare il metodo su quest'ultimo, stampare in bella forma il nome del metodo e il valore restituito, e restituire il valore stesso.

Provate a sostituire qualche oggetto in un programma che avete già scritto (purché assegnato a una variabile che ha per tipo un'interfaccia) con un oggetto generato da `makeDebuggable()` e guardate che cosa succede. Notate che per poter accedere a `makeDebuggable()` da un altro progetto dovete andare in *Project/Properties/Build Path/Projects* e selezionare il progetto che contiene la classe `Debug`.