Sul pattern Singleton

1 Introduzione

In molte situazioni è necessario che venga istanziato un solo esemplare di una data classe. Per esempio: se si ha un riproduttore di file musicali sarà bene che esso venga istanziato una sola volta per non ritrovarsi due riproduzioni contemporanee; uno spooler di stampa dovrebbe tenere una coda unica anche se ci sono più stampanti attive; il manager del file system dovrebbe essere unico, come pure una cache; ecc.

Come si fa a garantire che di una classe non possa essere istanziato più di un oggetto?
Si tratta anzitutto di rendere impossible l'uso del costrutto `new` da parte del programma utente e di fornire un metodo indiretto per ottenere una istanza (l'unica) della classe.

A tal fine occorre:

- dichiarare privato il costruttore, in modo che esso possa essere visto solo dall'interno della classe Singleton e non dal programma utente (ciò rende impossibile l'istanziazione di un oggetto dall'esterno della classe Singleton);
- prevedere il metodo (pubblico) *statico* e cioè di classe, in modo che esso sia comunque visibile. Questo metodo deve istanziare un esemplare se ciò non è ancora accaduto, oppure restituire l'oggetto già istanziato in precedenza senza istanziare ulteriori esemplari.

La classe Singleton prende quindi questa forma

```
public class Singleton {
    private static Singleton instance = null    //all'inizio non c'è
    private static String nome= "XX";           //tanto per dargli un nome

    public static Singleton getInstance() {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }

    private Singleton() {           //costruttore privato !!
    }

    public String getNome(){        //restituisce il nome
        return nome;
    }
}
```

Nel programma utente si avrà:

```
Singleton unico;
::::
unico = Singleton.getInstance();
```

Definizione

> Il pattern Singleton garantisce che ci sia una sola istanza di una data classe e fornisce un punto di accesso globale a tale istanza.

2 Osservazioni

Anzitutto conviene osservare che è possibile anche prevedere che la classe Singleton possa permettere l'istanziazione di un numero controllato di esemplari, per esempio 3. A tal fine basta opportunamente cambiare il metodo `getInstance()`.

Il Pattern, per quanto semplice presenta alcuni trabocchetti:

- In un sistema multithreading possono esserci due thread che tentano di istanziare un singleton. Consideriamo questo frammento di codice:

```
public static Singleton getInstance() {
    if (instance == null)
                      // Questo è il punto !!
        instance = new Singleton();
    return s;
}
```

  È possibile che il primo thread venga fermato dopo che ha effettuato il test e prima dello statement di assegnamento. Se ora corre il secondo thread, questo trova che nessun esemplare è stato istanziato e quindi l'istanziazione a luogo. Successivamente, quando il primo thread viene riattivato, avendo il test dato effetto negativo, si ha una nuova istanziazione. In sostanza vengono creati 2 esemplari.

  Il problema si risolve rendendo il metodo getInstance() sincronizzato:

```
public static synchronized Singleton getInstance() {
```

- Un problema analogo si verifica in sistemi dove le classi (gli oggetti) vengono caricati dinamicamente e usati finché servono (così fanno le Applet). Il Garbage Collector provvede a eliminare gli oggetti che non servono più. Se uno di questi ha istanziato un Singleton, quanto il Garbage collector lo elimina, si perde il riferimento al singleton e quindi prima o poi anche l'oggetto singleton viene eliminato dal garbage collector, in quanto non viene più riferito. A questo punto una nuova chiamata a `getInstance()` produce effettivamente un nuovo oggetto.
  Il problema si evita assicurando che c'è un tratto di codice (per esempio un thread che resta sempre presente) che non viene mai eliminato dal Garbage Collector e che mantiene il riferimento al singleton. In altre parole, deve restare un riferimento all'oggetto.

- Da ultimo osserviamo che il costruttore privato deve comunque essere dichiarato, altrimenti Java lo introduce di default in forma pubblica.