Il pattern Singleton

a cura di **Angelo Furfaro**

da “Design Patterns”, Gamma et al.

“Patterns in Java”, Grand

Dipartimento di
Ingegneria Informatica, Elettronica, Modellistica e Sistemistica
Università della Calabria, 87036 Rende(CS) - Italy
Email: a.furfaro@unical.it
Web: http://angelo.furfaro.dimes.unical.it

Singleton

Classificazione

- Scopo: creazionale
- Raggio d'azione: basato su oggetti

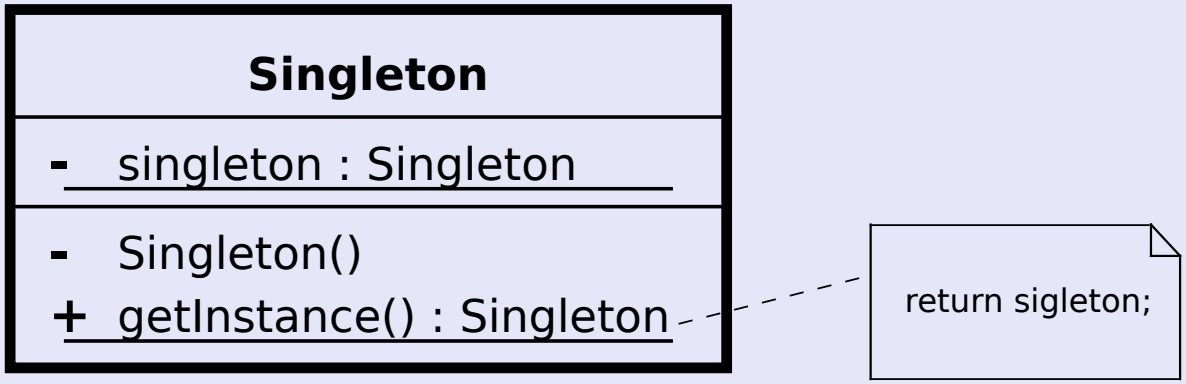
Scopo

- Assicurare che una classe abbia una sola istanza e fornire un punto di accesso globale a tale istanza.

Motivazione

- È importante poter assicurare che per alcune classi esista una sola istanza.
- In un sistema potrebbero esistere più stampanti, ma potrebbe essere presente soltanto una coda di stampa. In un sistema dovrebbe essere presente solo un file system e un solo window manager.
- Per assicurare che una classe abbia una sola istanza e che tale istanza sia facilmente accessibile per gli utilizzatori si può fare in modo che la classe stessa abbia la responsabilità di creare le proprie istanze.
- La classe può assicurare che nessun'altra istanza possa essere creata e può fornire un modo semplice per accedere all'istanza.

Struttura



Applicabilità

- Quando deve esistere esattamente un'istanza di una classe e tale istanza deve essere resa accessibile ai client attraverso un punto di accesso noto a tutti gli utilizzatori.
- Quando l'unica istanza deve poter essere estesa attraverso la definizione di sottoclassi ed i client devono essere in grado di utilizzare le istanze estese senza dover modificare il proprio codice.

Singleton in Java

- Per impedire l'istanziazione arbitraria di oggetti tutti costruttori della classe devono essere privati.
- L'accesso all'unica istanza avviene tramite un metodo statico (di classe) `getInstance()` il quale nasconde i dettagli relativi al processo di creazione.
- Nell'esempio si utilizza la tecnica della *lazy initialization*.
- Il metodo `getInstance()` è reso `synchronized` per garantire l'atomicità del processo di creazione nel caso di accesso concorrente alla classe.

```
public final class Singleton{
 ...
  private static Singleton INSTANCE=null;
  private Singleton(){}

  public static synchronized Singleton getInstance(){
    if ( INSTANCE==null ){
      INSTANCE=new Singleton();
    }
    return INSTANCE;
  }
 ...
}//Singleton
```

Singleton in Java

Problemi con la serializzazione

- Se la classe `Singleton` è serializzabile, per garantire che non siano create istanze leggendo da un flusso occorre aggiungere il metodo `readResolve()` come segue:

```
public final class Singleton implements Serializable{
 ...
  private static Singleton INSTANCE=null;
  private Singleton(){}

  public static synchronized Singleton getInstance(){
     if ( INSTANCE==null ){
       INSTANCE=new Singleton();
     }
     return INSTANCE;
  }
  private Object readResolve(){
      return getInstance();
  }
 ...
}//Singleton
```

- Tuttavia questa soluzione non risolve completamente il problema (Effective Java Item 77)

Utilizzo delle enumeration

- La soluzione migliore per implementare il Singleton in Java è ricorrere al costrutto `enum`.
- Si risolvono automaticamente tutti i problemi di accesso concorrente e quelli legati alla serializzazione.
- In tal caso si accede direttamente all'istanza (`Singleton.INSTANCE`).

```
public enum Singleton{
    INSTANCE;
    ...
}
```