Programmazione Orientata agli Oggetti in Linguaggio Java

Design Pattern:
Storia
Parte b

versione 2.1

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Una Visione Critica dei Pattern Originali
- Il Pattern Singleton
- Il Pattern Factory Method

I Pattern Originali Dopo 10 Anni

- Dopo 10 anni
  - ⇨alla conferenza OOPSLA 2004, una tavola rotonda sui pattern della GoF 10 anni dopo
- Obiettivo
  - ⇨verificare quali tra i pattern originali si sono realmente confermati come soluzioni efficaci nel tempo
  - ⇨e quali, viceversa, avrebbero potuto essere omessi

I Pattern Originali Dopo 10 Anni

- Il responso
  - ⇨alcuni pattern da rimuovere
  - ⇨alcuni pattern discutibili
- Pattern da rimuovere
  - ⇨Bridge, Interpreter
- Pattern discutibili
  - ⇨Singleton, Chain of Responsibility (poco usato)

Nome: Singleton
Categoria: creazionale
Difficoltà di apprendimento limitata
Difficoltà di applicazione media

Il Pattern Singleton

- Un esempio di pattern discusso
  - ⇨Singleton
  - ⇨è un pattern di tipo “creazionale”
- Descrizione del pattern
  - ⇨serve a creare una classe con un’unica istanza, che rappresenta un oggetto con visibilità “globale”, accessibile in tutta l’applicazione

Il Pattern Singleton

>> volpiEConigli2 – Configurazione.java

- Esempio
  - ⇨volpi e conigli
  - ⇨per ciascuna esecuzione c’è un’unica configurazione della simulazione
  - ⇨è opportuno che la configurazione sia visibile da molte classi (Gioco, Volpe, Coniglio, Test) >> visibilità globale
- In questo caso
  - ⇨è possibile applicare il singleton

```
package it.unibas.volpieconigli2.modello;
public class Configurazione {

   private int numeroRighe;
   private int numeroColonne;
   ...

   private static Configurazione singleton = new Configurazione();

   private Configurazione() {}

   public static Configurazione getInstance() {
      return Configurazione.singleton;
   }

   public void carica(String nomeFile) throws Exception {
      java.util.Properties proprieta = caricaProperties(nomeFile);
      ...
   }

   public int getNumeroInizialeConigli() {
      return (int)(this.numeroRighe * this.numeroColonne * this.percentualeConigli);
   }
   ...

}
```

il riferimento all'unica istanza della classe viene mantenuto come valore della proprietà statica singleton
es: singleton vale #2367

il costruttore è privato; non è possibile creare oggetti all'esterno

il metodo getInstance() restituisce sempre lo stesso riferimento (#2367)

Il Pattern Singleton

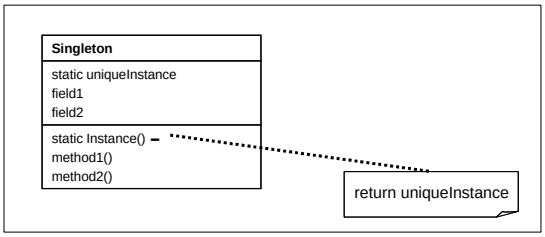
- In tutte le altre classi
  - ⇨posso ottenere il riferimento al singleton attraverso la chiamata
    Configurazione.getInstance()
  - ⇨e poi chiamarne i metodi; es:

    Configurazione.getInstance().getNumeroRighe();

    Configurazione.getInstance().getFameLimiteVolpi();

Il Pattern Singleton

Singleton
Diagramma UML della GoF



fonte: Design Patterns – Elements of Reusable OO Code

Il Pattern Singleton

- Perchè singleton è discutibile ?
  - ⇨per via di due problemi
- Problema n. 1
  - ⇨utilizzo esagerato
  - ⇨è opportuno usare un singleton solo nei casi in cui serve effettivamente un componente unico e globale
  - ⇨ma questa scelta non deve essere frettolosa

Il Pattern Singleton

- Esempio: scacchiera
  - ⇨potrebbe essere un singleton ?
- In generale no
  - ⇨non è indispensabile la visibilità globale
  - ⇨che deve essere, per quanto possibile, impedita
  - ⇨altrimenti altri componenti potrebbero per errore modificare lo stato della scacchiera

Il Pattern Singleton

- In altri termini
  - ⇨per poter essere un singleton, un componente deve essere effettivamente condivisibili a livello globale
  - ⇨ovvero tale per cui, pur essendo condiviso tra vari client, questi non possano interferire tra di loro modificandone lo stato
  - ⇨nel caso di configurazione questo rischio è molto basso perchè i metodi set sono privati
  - ⇨un altro esempio plausibile di singleton: Console

Il Pattern Singleton

- Ma in effetti, a pensarci bene
  - ⇨la stessa configurazione potrebbe non essere per sempre un singleton
  - ⇨se mettessi in linea l'applicazione su un sito Web, potrei avere vari giocatori che giocano, ciascuno con la sua configurazione
  - ⇨a quel punto il singleton non andrebbe più bene

Il Pattern Singleton

- Problema n. 2
  - ⇨quando è realmente utile il singleton ?
- Infatti
  - ⇨si tratta un componente "unico" visibile in tutta l'applicazione
- Ma...
  - ⇨questo è equivalente ad una classe con costruttore privato e metodi statici

Il Pattern Singleton

- In effetti
  - ⇨nella versione originale, Configurazione era un componente statico
  - ⇨esattamente come it.unibas.utilita.Console
- Di conseguenza
  - ⇨in molti casi l'utilizzo di un singleton può essere surrogato utilizzando viceversa un componente statico

Il Pattern Singleton

- Quando serve realmente il singleton?
  - ⇨ci sono due casi in cui è indispensabile
- I caso
  - ⇨ho bisogno di un oggetto, ovvero di manipolare un riferimento
- II caso
  - ⇨il componente unico deve avere comportamento polimorfo (alcuni dei metodi devono essere sovrascritti nelle sottoclassi)

Il Pattern Singleton

- Un esempio del I caso
  - ⇨AnimaleNullObject in volpi e conigli
- Infatti
  - ⇨costruire tanti oggetti di tipo animale nullo è inutile, visto che si comportano tutti allo stesso modo
  - ⇨mi basta un componente unico e globale
  - ⇨ma deve essere necessariamente un oggetto perchè devo disporlo sulla scacchiera

```
package it.unibas.volpieconigli2.modello;

public class AnimaleNullObjectSingleton {

    private static AnimaleNullObjectSingleton singleton =
                                        new AnimaleNullObjectSingleton();

    private AnimaleNullObjectSingleton() {}

    public static AnimaleNullObjectSingleton getInstance() {
        return singleton;
    }

    public boolean isConiglio() { return false; }

    public void agisci(Scacchiera scacchiera) { return; }

    public String toString() { return "    "; }

}
```

Il Pattern Singleton

- In questo caso
  - ⇨cambia l'inizializzazione della scacchiera

```
public Scacchiera(int numeroRighe, int numeroColonne) {
        elementi = new Animale[numeroRighe][numeroColonne];
        for (int i = 0; i < numeroRighe; i++) {
            for (int j = 0; j < numeroColonne; j++) {
                elementi[i][j] = AnimaleNullObjectSingleton.getInstance();
            }
        }
}
```


Il Pattern Singleton

- Un esempio del II caso
  - ⇨in Java non esiste
  - ⇨dal momento che una classe basata su singleton non può essere estesa
- Infatti
  - ⇨se il costruttore è privato la classe diventa automaticamente finale

Il Pattern Singleton

- La soluzione del libro originale
  - ⇨dichiarare il costruttore protected
  - ⇨ma anche questo in Java non ha senso, perchè non si tratta più di un singleton (se il costruttore è protected, altre classi dello stesso package possono creare oggetti)
- Viceversa
  - ⇨la soluzione originale con costruttore protected aveva senso in C++

Il Pattern Singleton

- Di conseguenza
  - ⇨in Java, l'ambito di applicabilità del singleton è molto limitato
  - ⇨ha senso solo in casi simili a quello di AnimaleNullObject
  - ⇨bisogna evitare di applicarlo quando non è indispensabile

Riassumendo

- Creational patterns (5)
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton (discutibile, visto)
- Structural Patterns (7)
  - Adapter
  - Bridge (da rimuovere)
  - Composite
  - Decorator
  - Facade (visto)
  - Flyweight
  - Proxy
- Behavioral Patterns (11)
  - Chain of Responsibility (discutibile)
  - Command
  - Interpreter (da rimuovere)
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor
- Altri pattern
  - Null Object (visto)
  - DAO (visto)

Riassumendo

- Una Visione Critica dei Pattern Originali
- Il Pattern Singleton
- Il Pattern Factory Method

Termini della Licenza