Programmazione Orientata agli Oggetti in Linguaggio Java

Design Pattern:
Storia
Parte a

versione 2.1

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

Sommario

- Definizione e Origine
- Il Libro “Design Patterns” della GoF
- Il Pattern Facade
- Evoluzione dei Pattern
  - ⇨Il Pattern Null Object
  - ⇨Il Pattern DAO

Definizione e Origine

- Design Pattern
  - ⇨letteralmente “motivi ricorrenti di progettazione”
  - ⇨si tratta di modelli di soluzioni
  - ⇨per problemi ricorrenti di sviluppo del software
  - ⇨consolidatesi nel tempo attraverso l’esperienza
  - ⇨con i linguaggi ad oggetti

sono “scheletri” di soluzione, sufficientemente generali da essere riadattabili a vari contesti applicativi

si riferiscono a problemi che si presentano con frequenza

nascono dall’ esperienza

sono basati su tecnologie OO

Definizione e Origine

- In effetti
  - ⇨l’origine non è legata al software, ma all’architettura
- Il primo ideatore
  - ⇨Cristopher Alexander
  - ⇨“*A Pattern Language: Towns, Buildings, Construction*”, Oxford University Press, 1977
  - ⇨circa 250 pattern diversi per l’architettura degli edifici

Definizione e Origine

- La definizione di Alexander
  - ⇨"*Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice*"

Definizione e Origine

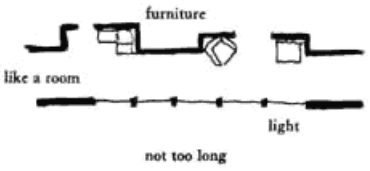
- La descrizione di un pattern
  - ⇨un breve nome
  - ⇨una descrizione del problema da risolvere
  - ⇨una descrizione della soluzione suggerita
  - ⇨una discussione del contesto di applicazione, ovvero delle conseguenze, dei vantaggi e degli svantaggi collegati all'introduzione del pattern

Definizione e Origine

- Un esempio di pattern in architettura
- Nome: “Corridoio corto”
- Problema:
  - ⇨“... i corridoi lunghi e inutili rappresentano le cose peggiori dell’architettura moderna..”

Definizione e Origine

- Soluzione
  - ⇨“realizzare corridoi brevi, rendendoli per quanto possibile simili a stanze con tappeti, mobili, librerie”
  - ⇨“prevedere dimensioni ampie e abbondante illuminazione”



Definizione e Origine

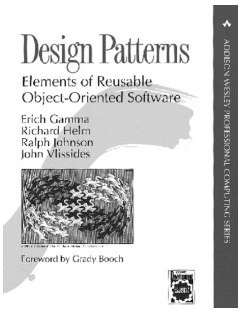
- L'idea di Alexander
  - costruire edifici utilizzando esclusivamente pattern
  - ovvero definire un "linguaggio di pattern" per la progettazione architettonica
- Pattern Language
  - insieme di pattern collegati tra di loro che collaborano a costruire un intero artefatto

Il Libro "Design Patterns" della GoF

- I pattern nella programmazione
  - nascono con un libro classico sull'argomento della cosiddetta "Gang of Four" (GoF)
- La Gang of Four
  - Erich Gamma, Software Techn. Center, Svizzera
  - Richard Helm, IBM, Sydney
  - Ralph Johnson, Univ. of Illinois, USA
  - John Vlissides, IBM John Watson Center, USA

Il Libro "Design Patterns" della GoF

- Il libro
  - ⇨"*Design Patterns – Elements of Reusable Object-Oriented Software*", Addison Wesley, ottobre 1994
- Attualmente
  - ⇨oltre la 25ma ristampa



Il Libro "Design Patterns" della GoF

- Nel libro
  - ⇨viene introdotto il concetto di design pattern applicato allo sviluppo del software
  - ⇨vengono illustrati i principi metodologici alla base dell'uso del pattern
  - ⇨viene fornito un elenco di 23 pattern divisi in tre categorie
  - ⇨i linguaggi di riferimento sono il C++ e Smalltalk (i linguaggi di riferimento all'epoca)

Il Libro "Design Patterns" della GoF

- Principio metodologico fondamentale
  - ⇨sviluppare per consentire il cambiamento
- Principio n. 1
  - ⇨programmare con le interfacce e non con le implementazioni
- Principio n. 2
  - ⇨favorire l'associazione rispetto all'ereditarietà
- Principio n. 3
  - ⇨separare le cose che possono cambiare da quelle che presumibilmente resteranno le stesse

Il Libro "Design Patterns" della GoF

- Altri principi importanti
  - ⇨non espliciti ma scontati
- Principio n. 4
  - ⇨rendere il codice efficiente (es: evitare la creazione eccessiva di oggetti)
- Principio n. 5
  - ⇨mantenere il codice manutenibile (es: evitare troppe classi, concentrare il codice ecc.)

Il Libro "Design Patterns" della GoF

- Il catalogo dei pattern della GoF
  - ⇨23 pattern divisi in tre categorie
- Creational patterns
  - ⇨riguardano la creazione degli oggetti
- Structural Patterns
  - ⇨riguardano l'organizzazione degli oggetti
- Behavioral Patterns
  - ⇨riguardano il comportamento degli oggetti

Il Libro "Design Patterns" della GoF

- Creational patterns (5)
  - ⇨ Abstract Factory
  - ⇨ Builder
  - ⇨ Factory Method
  - ⇨ Prototype
  - ⇨ Singleton
- Structural Patterns (7)
  - ⇨ Adapter
  - ⇨ Bridge
  - ⇨ Composite
  - ⇨ Decorator
  - ⇨ Facade
  - ⇨ Flyweight
  - ⇨ Proxy
- Behavioral Patterns (11)
  - ⇨ Chain of Responsibility
  - ⇨ Command
  - ⇨ Interpreter
  - ⇨ Iterator
  - ⇨ Mediator
  - ⇨ Memento
  - ⇨ Observer
  - ⇨ State
  - ⇨ Strategy
  - ⇨ Template Method
  - ⇨ Visitor

Il Pattern Facade

Nome: Facade
Categoria: strutturale
Difficoltà di apprendimento limitata
Difficoltà di applicazione limitata

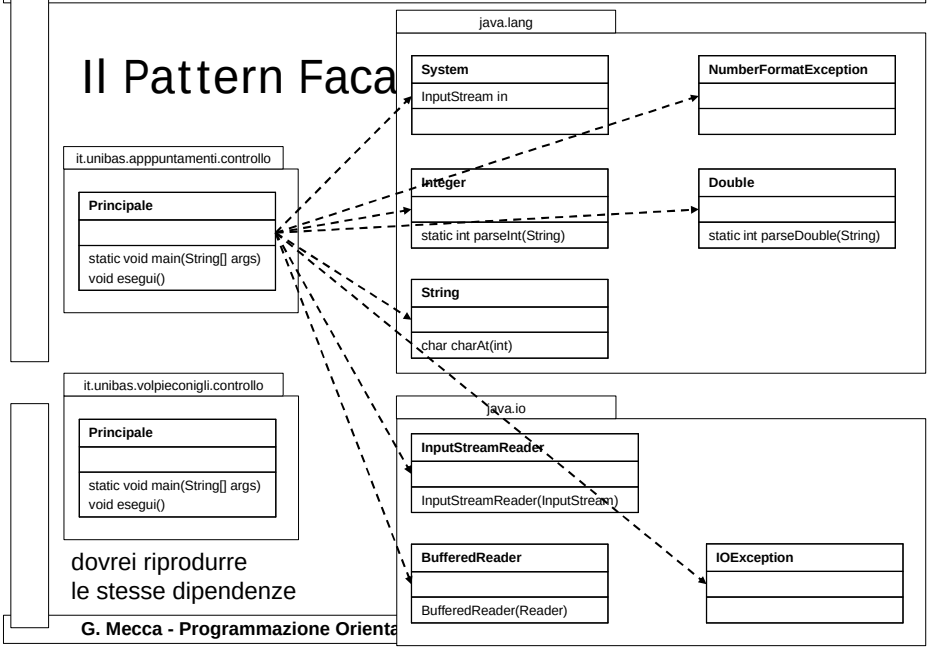
- Un esempio di pattern
  - ⇨Facade
  - ⇨è un pattern di tipo “strutturale”
- Descrizione del pattern
  - ⇨serve a creare una singola interfaccia semplificata per l’accesso ad un insieme di interfacce e classi più complicato

Il Pattern Facade

- Un esempio
  - ⇨il problema della lettura dallo standard input in Java
  - ⇨richiede l’accesso a svariate classi delle API per la creazione del flusso ed in particolare: System, InputStream, InputStreamReader, BufferedReader, IOException ...
  - ⇨e ad altre classi per l’input formattato: Integer, Double, Float, Boolean, NumberFormatException ...

Il Pattern Facade

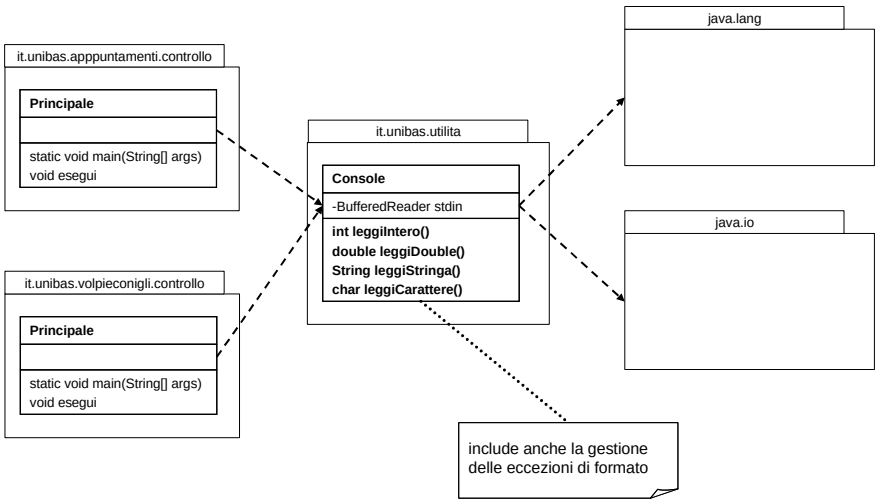
- In teoria
  - ⇨sarebbe possibile scrivere una classe che faccia standard input direttamente accedendo ai metodi di tutte queste classi
  - ⇨es: it.unibas.appuntamenti.controllo.Principale
  - ⇨il programmatore dovrebbe conoscere tutte le classi necessarie e scrivere opportunamente il codice delle chiamate dei metodi



Il Pattern Facade

- La classe it.unibas.utilita.Console
  - ⇨fornisce una “facciata” unica per il complesso di classi necessarie
  - ⇨quattro metodi semplici già pronti
  - ⇨int leggiIntero(), double leggiDouble(), String leggiStringa(), char leggiCarattere()
  - ⇨include la gestione delle eccezioni
  - ⇨nasconde la complessità delle classi retrostanti

Il Pattern Facade



Il Pattern Facade

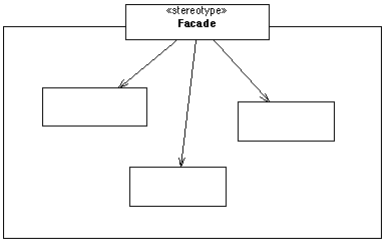
- In generale
  - una Facade è una classe che fornisce, attraverso una interfaccia singola, accesso semplificato ad una collezione di interfacce complesse
  - è uno strumento per organizzare l'applicazione in "sottosistemi", ciascuno dei quali accessibile dall'esterno in maniera semplificata attraverso la facciata
  - es: sottosistema "lettura dallo standard input"

Il Pattern Facade

- Di conseguenza
  - in alcuni casi le classi "client" possono semplicemente usare l'interfaccia semplificata della facciata (la maggioranza)
  - in altri casi potrebbe essere necessario accedere alle singole classi nascoste
- Altri esempi di Facade
  - it.unibas.utilita.Logger
  - la classe System.Date di C#

Il Pattern Facade

Facade
Diagramma UML della GoF



fonte: http://www.tml.hut.fi/~pnr/Tik-76.278/gof/html/

Il Pattern Facade

- Una prima annotazione sui nomi
  - ⇨è convenzione riportare nel nome di una classe sviluppata secondo il pattern il nome del pattern utilizzato
  - ⇨per renderne più evidente l'utilizzo
- Di conseguenza
  - ⇨la classe Console avrebbe dovuto chiamarsi "ConsoleFacade"

Il Pattern Facade

- Una seconda annotazione
  - ⇨dall'esempio dovrebbe essere evidente la differenza tra pattern e framework
- Pattern
  - ⇨scheletro di soluzione per un problema in un certo contesto
- Framework
  - ⇨collezione di classi e di regole riutilizzabili per lo sviluppo di una classe di applicazioni

Il Pattern Facade

- Quindi
  - ⇨i framework sono fatti anche di codice, i pattern no
  - ⇨i framework sono legati ad una classe specifica di applicazioni (es: test, collezioni, programmazione grafica ecc.), i pattern sono legati ad una classe di problemi di programmaizone

Evoluzione dei Pattern

- Dopo il libro della “Gang of Four”
  - ⇨si crea molto interesse e il movimento dei design pattern decolla rapidamente
  - ⇨vengono suggeriti molti altri pattern
  - ⇨nascono vari convegni internazionali
  - ⇨es: PLoP, Pattern Language of Programming, nel 2004 all’11 edizione
  - ⇨nascono altri concetti collegati; es: antipattern: un errore da evitare

Nome: NullObject
Categoria: --
Difficoltà di apprendimento limitata
Difficoltà di applicazione limitata

Il Pattern Null Object

- In effetti
  - ⇨esistono pattern che non fanno parte del catalogo della GoF perchè sono successivi
  - ⇨ma che nella pratica si rivelano molto utili
- Un esempio
  - ⇨il pattern Null Object (“Oggetto Nullo”)
- Descrizione del pattern
  - ⇨un modo per sostituire un riferimento nullo con un oggetto che non fa nulla

Il Pattern Null Object

- Esempio
  - ⇨nell'applicazione volpi e conigli, solo alcune posizioni sulla scacchiera sono occupate da animali
  - ⇨i restanti riferimenti hanno valore null
- Di conseguenza
  - ⇨ogni volta che si preleva un riferimento dalla scacchiera è necessario, prima di fare qualsiasi cosa, verificare che non sia null

Il Pattern Null Object

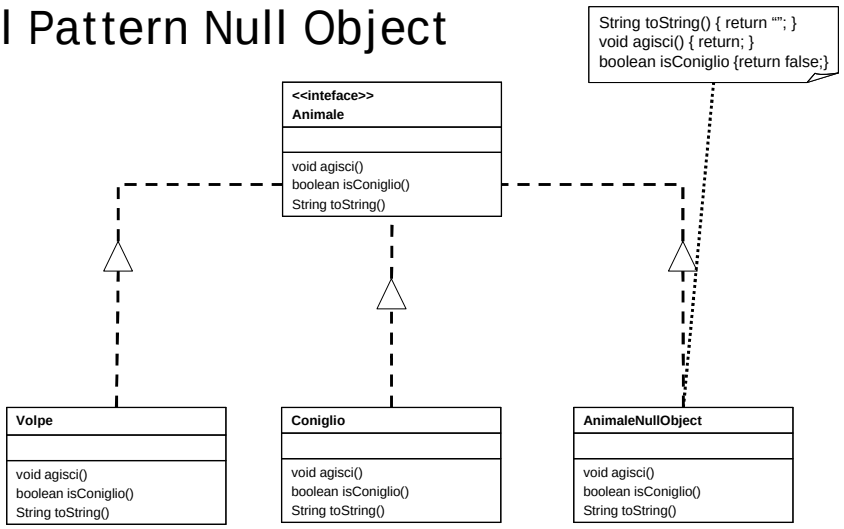
```
public void simula(Scacchiera scacchiera) {
    for (int i = 0; i < scacchiera.getRighe(); i++) {
        for (int j = 0; j < scacchiera.getColonne(); j++) {
            Animale animale = scacchiera.getElemento(i, j);
            if (animale != null) {
                animale.agisci(scacchiera);
            }
        }
    }
}
```

lo stesso tipo di test
è necessario ogni volta che prelevo
un animale sulla scacchiera
es: stampa della scacchiera

Il Pattern Null Object

- Una possibile soluzione
  - ⇨definisco una classe AnimaleNullObject
  - ⇨che implementa l'interfaccia animale
  - ⇨ma i cui metodi corrispondono ad un comportamento neutro
- In particolare
  - ⇨agisci() non effettua nessuna operazione
  - ⇨toString() restituisce la stringa vuota

Il Pattern Null Object



Il Pattern Null Object

- In questo caso
  - ⇨posso inizializzare la scacchiera invece che attraverso riferimenti null, riempendola di riferimenti ad animali nulli
  - ⇨da quel momento in poi non è più necessario verificare se i riferimenti prelevati dalla scacchiera sono nulli o meno

Il Pattern Null Object

```
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] = new AnimaleNullObject();
         }
      }
}

public void simula(Scacchiera scacchiera) {
      for (int i = 0; i < scacchiera.getRighe(); i++) {
         for (int j = 0; j < scacchiera.getColonne(); j++) {
            scacchiera.getElemento(i, j).agisci();
         }
      }
}
```

Il Pattern Null Object

- In generale
  - ⇨il pattern NullObject è utile in tutti i casi in cui riferimenti ad oggetti di un certo tipo devono essere mischiati con riferimenti nulli
  - ⇨consente di evitare l'utilizzo di riferimenti nulli e quindi le verifiche corrispondenti
  - ⇨evita, di conseguenza, errori del tipo NullPointerException

Il Pattern DAO

Nome: DAO
Categoria: --
Difficoltà di apprendimento limitata
Difficoltà di applicazione limitata

- Un ulteriore esempio
  - ⇨il pattern DAO ("Data Access Object")
  - ⇨un pattern per la gestione della persistenza
- Descrizione del pattern
  - ⇨un oggetto specializzato in operazioni di persistenza per una classe di oggetti del modello
  - ⇨è in grado di caricare, salvare e modificare oggetti di una classe in memoria persistente

Il Pattern DAO

- Vantaggi del DAO
  - ⇨aumenta la coesione e consente di modularizzare il codice (separa lo strato del modello da quello della persistenza)
  - ⇨riduce l'accoppiamento perchè isola lo strato del controllo dalla tecnologia della persistenza (il controllo interagisce esclusivamente con i DAO)

Evoluzione dei Pattern

- Attualmente
  - ⇨esistono sulla rete molti cataloghi di pattern
  - ⇨in cui si contano centinaia di pattern diversi
  - ⇨anche in contesti applicativi vari (es: sviluppo di applicazioni distribuite)
- Esempi
  - ⇨http://hillside.net/patterns/
  - ⇨http://home.earthlink.net/~huston2/dp/patterns.html

Evoluzione dei Pattern

- La diffusione dei pattern
  - ha aspetti senza dubbio positivi
  - i pattern migliorano la comunicazione tra i programmatori (“usiamo una Facade qui..”)
  - i pattern migliorano la documentazione del software (alcuni pezzi dei diagrammi diventano autoesplicativi)
  - i pattern favoriscono la diffusione delle esperienze

Evoluzione dei Pattern

- Ma ha anche aspetti negativi
  - una forte proliferazione è contraria al concetto stesso di pattern
  - perchè si scopra un pattern è necessario individuare una soluzione efficace ad un problema ricorrente (quindi i pattern vengono scoperti lentamente)
  - inoltre, non sempre i propositori sono del tutto autorevoli

Evoluzione dei Pattern

- Una degenerazione del movimento
  - ⇨riapplicare anche al contesto del software l'idea del "linguaggio di pattern"
  - ⇨ovvero regole per descrivere la struttura di un'applicazione esclusivamente in termini di pattern
- Ma...
  - ⇨questo è contrario allo spirito del libro della GoF

Evoluzione dei Pattern

- Infatti
  - ⇨nel libro vengono chiaramente discusse le differenze con i pattern di Alexander
  - ⇨l'architettura esiste da molti più anni del software, le soluzioni architetturali sono molto più consolidate, il concetto di linguaggio ha senso
  - ⇨viceversa, il software è ancora immaturo per spingere i pattern a diventare una soluzione definitiva
  - ⇨ci vorranno anni per consolidare le tecniche

Riassumendo

- Definizione e Origine
- Il Libro “Design Patterns” della GoF
- Il Pattern Facade
- Evoluzione dei Pattern
  - ⇨Il Pattern Null Object
  - ⇨Il Pattern DAO

Termini della Licenza