Programmazione Concorrente e Distribuita 2014-2015. Prima parte.

Andrea Giacomo Baldan 579117

September 4, 2015

Contents

1 Organizzazione e scelte implementative

Pur trattandosi della prima parte del progetto e quindi di un programma non particolarmente complesso, ho comunque deciso di implementare il codice con un minimo di organizzazione e modularità seguendo un design pattern che potesse facilitare l'estensione futura del software per le parti successive. Anche in assenza di un interfaccia grafica, ho scelto di seguire il design pattern MVC per la modularità e la separazione logica - output che offre, inoltre nel caso ipotetico in cui venisse implementata in futuro una GUI, il design pattern scelto agevolerebbe notevolmente il lavoro.
La directory src contenente i sorgenti, è dunque suddivisa nelle tre classiche sotto-directory models, views, controllers, dove rispettivamente:

- models contiene gli oggetti che rappresentano il puzzle e supporti I/O per l'interazione con i file di input e output
- views contiene le classi adibite alla rappresentazione output dei risultati
- controllers contiene le classi che si occupano della logica da applicare ai models

1.1 Organizzazione delle classi

Le classi sono raggruppate in un package, **puzzlesolver** con all'esterno la classe **PuzzleSolver** che contiene il *main* e le chiamate esecutive del programma.

```
src
|__ models
|    |___ Puzzle.java
|    |___ Piece.java
|    |___ IOPuzzle.java
|    |___ IOFile.java
|    |___ IPiece.java
|
|__ views
|    |___ PuzzleView.java
|    |___ PuzzleView.java
|
|__ controllers
|    |___ SortAlg.java
|    |___ SortAlgSeq.java
|    |___ PuzzleController.java
|    |___ IPuzzleController.java
|
|__ PuzzleSolver.java
```

1.1.1 Models

La directory models contiene essenzialmente 3 oggetti distinti:

- La gerarchia **IOFile > IOPuzzle:** Entrambe formano l'oggetto incaricato della gestione dell'input e output del programma, in questo caso mediante files, IOPuzzle.java estende la classe base astratta IOFile.java implementando i metodi *read* e *write* secondo le specifiche suggerite dalla consegna.
- **IPiece e Piece:** IPiece.java fornisce un interfaccia che delinea la "forma" di un generico pezzo del puzzle ordinabile, Piece.java è l'implementazione di tale interfaccia, possiede anche l'override del metodo *toString* per facilitare la formattazione output richiesta dalle specifiche.
- **Puzzle:** Rappresenta l'oggetto puzzle formato da un insieme di pezzi (IPiece) raggruppati mediante un contenitore, in questo caso è stato scelto i Vector per la semplicità di gestione che fornisce. Puzzle contiene tutti i metodi di gestione generici del puzzle.

1.1.2 Views

La directory views contiene le classi atte a visualizzare l'output, o in questo caso a fornire un livello di astrazione con la parte di programma che si occupa dell'output (IOPuzzle), contiene solamente un interfaccia, **IPuzzle-**

View.java che delinea il comportamento della vista e la sua implementazione **PuzzleView.java**.
Questa classe formata da un campo dati di tipo **IOFile** viene utilizzata all'interno delle classi controller e garantisce un certo grado di sicurezza permettendo interazione con i modelli solamente mediante i metodi pubblici forniti.

1.1.3 Controllers

Qui risiedono le classi adibite alla parte logica del programma, **SortAlg.java** è la classe base astratta che rappresenta l'algoritmo di ordinamento del puzzle, estesa da **SortAlgSeq.java** che, come suggerisce il nome, implementa il metodo astratto *sort*, mentre **IPuzzleController.java** e la sua implementazione **PuzzleController.java** costituiscono l'effettivo controller a disposizione dell'utente per l'ordinamento del puzzle.

1.2 Principi di OOP

1.2.1 Information hiding

I campi dati delle classi sono stati dichiarati tutti *private*, accedibili mediante classici metodi *getters* e *setters*. Tutti i metodi di utilità all'interno della classe SortAlgSeq sono stati dichiarati *private*, questo perchè l'organizzazione mediante classe base astratta permette di ridefinire a piacimento il metodo di ordinamento scelto, di cui i suddetti metodi fanno parte.

1.2.2 Polimorfismo

Al fine di garantire un buon livello di estendibilità, il puzzle è stato pensato in modo da operare con oggetti di tipo **IPiece**, ovvero l'interfaccia che delinea la struttura e il comportamento dei pezzi del puzzle, in questo modo è possibile creare la propria "versione" dei pezzi senza dover modificare codice nella classe **Puzzle**.
Lo stesso si può affermare delle classi di gestione I/O, views e controller, l'algoritmo di ordinamento stesso, nocciolo della prima parte del progetto, poggia su una classe base astratta, ed è possibile estendere o creare altre versioni mediante override del metodo di ordinamento.

1.2.3 Modularità

La struttura secondo pattern MVC garantisce un buon livello di modularità fra le "parti" del programma, opportunamente organizzato nel package *puzzlesolver*.

2 Algoritmo di ricostruzione

L'algoritmo di risoluzione implementato nella classe SortAlgSeq, derivata dalla classe base astratta SortAlg, è appunto un algoritmo di risoluzione sequenziale, e si può riassumere in 3 passi:

1. Localizzazione del primo pezzo del puzzle, che corrisponde al pezzo avente "VUOTO" a nord e ad ovest, ciò avviene richiamando il metodo privato firstPiece.

2. Inizio ciclo: ordina la riga a partire dal primo pezzo mediante il metodo privato nextInRow.

3. Ricerca del pezzo a sud del primo pezzo della nuova riga ora ordinata, se il pezzo a sud non esiste e troviamo dunque "VUOTO", il ciclo si ferma in quanto tutte le righe sono quindi già state ordinate, altrimenti si ripete (1) utilizzando il "nuovo" primo pezzo. Di questa operazione se ne occupa il metodo privato nextInCol.

3 Note

Il progetto è stato sviluppato in ambiente linux, utilizzando la JVM versione 1.7.0. ed è stato testato sui computer del laboratorio Paolotti con esito positivo. E' stato creato un MakeFile apposito per la compilazione del progetto, è sufficiente lanciare il comando make. Con make clean vengono ripuliti i folder *bin* da tutti i file .class, è stato inoltre creato uno script bash per effettuare tutto il processo di pulizia-compilazione-esecuzione chiamato puzzlesolver.sh, richiede come parametri i due file rispettivamente di input e output.