Programmazione Concorrente e Distribuita 2014-2015. Prima parte.

Andrea Giacomo Baldan 579117

September 5, 2015

Contents

1 Modifiche finali

Le maggiori modifiche per l'ultima parte del progetto sono state apportate a livello organizzativo. Dovendo separare il lato client dal lato server, non sono state create classi aggiuntive, le esistenti sono però state "ridistribuite" in modo da rispettare le specifiche richieste; nella suddivisione **client** e **server** la gestione I/O è stata assegnato al lato client mentre la logica di ordinamento del puzzle è stata spostata al lato server. E' dunque compito del server ora occuparsi della gestione dei thread di ordinamento.

2 Implementazione

Le specifiche della terza parte del progetto prevedono la modifica della seconda parte al fine di ottenere un'applicazione distribuita mediante l'utilizzo del sistema RMI di java, differente dal classico metodo che utilizza i socket per connessioni remote.
Come si può notare dall'albero della directory *src*, il programma è stato suddiviso in due parti principali, con due distinti main.

2.1 Albero delle classi

```
src
|__ client
|    |___ controllers
|    |     |___ IPuzzleController.java
|    |     |___ PuzzleController.java
|    |
|    |___ models
|    |     |___ IOFile.java
|    |     |___ IOPuzzle.java
|    |     |___ IPiece.java
|    |     |___ Piece.java
|    |     |___ Puzzle.java
|    |
|    |___ views
|    |     |
|    |     |___ IPuzzleView.java
|    |     |___ PuzzleView.java
|    |
|    |___ PuzzleSolverClient.java
|
|__ server
     |___ controllers
     |     |___ IPuzzleServerController.java
     |     |___ PuzzleServerController.java
     |     |___ SharedSortStat.java
     |     |___ SortAlgFromBottom.java
     |     |___ SortAlgFromTop.java
     |     |___ SortAlg.java
     |
     |___ models
     |     |___ IPiece.java
     |     |___ Piece.java
     |     |___ Puzzle.java
     |
     |___ PuzzleSolverServer.java
```

2.2 Il server RMI

Per prima cosa è stata creata un interfaccia remota, è stato deciso di rendere accessibile il riferimento remoto al controller che si occupa dell'ordinamento del puzzle, pertanto *IPuzzleServerController* è l'interfaccia designata ad

estendere *Remote*, tutti i suoi metodo sono stati marcati *throws RemoteException* e la sua implementazione *PuzzleServerController* estende ora *UnicastRemoteObject* che permette di inserire il riferimento all'oggetto controller all'interno del registro RMI.
All'avvio del main lato server, viene creato l'oggetto *IPuzzleServerController* e viene inserito nel registro RMI mediante il metodo *Rebind*, da li rimane in attessa di eventuali connesioni da parte del client.

2.3 L'oggetto puzzle

Affinchè fosse possibile ordinare il puzzle mediante metodo remoto dal server, è stato necessario apportare modifiche anche alle classi model, sia lato server che lato client, questo perchè entrambe le parti necessitano di trasmettere l'oggetto puzzle; il client spedisce al server l'oggetto puzzle disordinato, il server risponde inviando l'oggetto puzzle ordinato. Ciò è reso possibile dall'interfaccia *Serializable*, che permette di serializzare appunto l'oggetto che la implementa e inviarne una copia all'oggetto remoto.
Sia *Piece* che *Puzzle* sono quindi un implementazione dell'interfaccia *Serializable*.

2.4 Il client

All'avvio del main lato client, vengono letti gli input forniti, e il metod *sort* della classe *PuzzleController* si occupa di ottenere il riferimento all'oggetto remoto (di tipo *IPuzzleServerController*) dal server mediante il metodo *Lookup* e richiama il metodo *sort* che risiede sul server, passando come parametro l'oggetto *Serializable* di tipo *Puzzle* da riordinare. Infine aggiorna l'oggetto *Puzzle* locale con la copia riordinata ottenuta in risposta dal server.

2.5 Robustezza

Nel caso di problemi di connessione, o caduta di una delle due parti durante una sessione di ordinamento viene notificata con il lancio di un eccezione remota o di tipo *ConnectException*, opportunamente gestite da blocchi *try/catch*.

3 Note

Per la compilazione, make dovrà avere come parametro la parte che si intende compilare, ovvero make Client per il client e make Server per il server, sono stati aggiunti i due script bash come richiesto da specifiche.
Non essendo espressamente richiesto nelle specifiche della terza parte, nel progetto non è stato implementato un sistema di gestione di richieste concorrenti da parte di clients multipli; nel caso la soluzione più rapida pensata consiste nell'inserire un contatore statico di client che richiedono connessione all'interno dell'oggetto remoto, un metodo di registrazione e una lista dei puzzle associati ad ogni client, o, in alternativa, un sistema di callback utilizzando un interfaccia remota e passando il riferimento del controller lato client al server, per evitare di scrivere sul registro rmi il riferimento al client servirebbe tuttavia l'utilizzo del compilatore rmi per generare stub e skeleton del riferimento da esportare sul server.