# Programmazione Concorrente e Distribuita 2014-2015. Prima parte.

### Andrea Giacomo Baldan 579117

## September 3, 2015

### **Contents**

1	Mod	lifiche finali	
2	Implementazione		
	2.1	Albero delle classi	
	2.2	Il server RMI	
	2.3	L'oggetto puzzle	
		Il client	

## 1 Modifiche finali

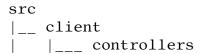
Le maggiori modifiche per l'ultima parte del progetto sono state apportate a livello organizzativo. Dovendo separare il lato client dal lato server, non sono state create classi aggiuntive, le esistenti sono però state "ridistribuite" in modo da rispettare le specifiche richieste; nella suddivisione **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



		IPuzzleController.java
		PuzzleController.java
         	           	models   IOFile.java   IOPuzzle.java   IPiece.java   Piece.java   Puzzle.java
		views
     	     	   IPuzzleView.java   PuzzleView.java
		PuzzleSolverClient.java
	serve	er
I—— `		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.