Relazione PCD - parte 3

Gabriele Pozzan
matricola 1051239

gennaio 2015

Il programma implementa un sistema client server per la risoluzione di un puzzle. Nello specifico il client invia al server le informazioni sui pezzi necessarie alla risoluzione e riceve la soluzione.

1 - Cambiamenti rispetto alla seconda parte

Sono stati effettuati alcuni cambiamenti nella struttura dei package e in alcuni dettagli implementativi delle classi, inoltre sono state aggiunte due classi e una interfaccia.

1.1 - Cambiamenti nei package

Le classi del programma sono state sistemate in tre package:

- **server**: contiene la classe *PuzzleSolverServer*
- **server.engine**: contiene la logica risolutiva del puzzle (come il package **engine** nelle parti precedenti)
- **client**: contiene la classe *PuzzleSolverClient*
- **common**: contiene le interfacce comuni a server e client, ovvero *Puzzle* e *PuzzleSolverServerIntf* e una nuova eccezione *WrongInput*

1.2 - Cambiamenti nell'implementazione

Il parametro in input del metodo *initialize* dell'interfaccia *Puzzle* (implementato poi in *AbstractPuzzle*) è stato cambiato da *Path* a *String*, questo perché in questa versione

distribuita del programma tale metodo sarà invocato remotamente e non avrebbe senso fornirgli un percorso sul file system della macchina client.
Anche il valore di ritorno del metodo *initialize* è stato cambiato da *boolean* a *String*, questo per poter fornire al client informazioni più precise nel caso di errore.

Il metodo *initialize* in questa versione dunque riceve la lista di pezzi sotto forma di stringa, a partire da questi prova a costruire un insieme di *PuzzlePiece* (come nelle parti precedenti), nel caso riscontri qualche errore termina l'esecuzione ritornando una stringa esplicativa (nelle versioni precedenti veniva semplicemente stampata a video) che verrà poi stampata dal client sul file di output (con un messaggio a video che inviterà ad andare a controllare cosa sia andato storto). In caso di successo il metodo ritornerà una stringa vuota.
La classe *Puzzle* è stata resa sottoclasse di *Remote* e la classe *AbstractPuzzle* sottoclasse di *UnicastRemoteObject* per permettere al client di ottenere un riferimento remoto ad un puzzle.

1.3 - Nuove classi e interfacce

Sono state aggiunte le seguenti classi e interfacce:

- **PuzzleSolverServerIntf**: interfaccia dell'oggetto server, fornisce il metodo *getPuzzle* che permette a un client di ottenere un riferimento a un puzzle remoto (e dunque di inizializzarlo e risolverlo in base ai propri dati in input)
- **PuzzleSolverServer**: implementa l'oggetto server, nel suo *main* crea un registro su localhost che accetta richieste sulla porta 2020 e vi carica un oggetto *PuzzleSolverServer*
- **PuzzleSolverClient**: implementa il client, in base al file indicato come input costruisce una stringa da fornire all'oggetto puzzle remoto, tenta poi di ottenere un riferimento remoto all'oggetto server (sempre su localhost:2020) e tramite quello tenta di ottenere un riferimento remoto a un puzzle
- **WrongInput**: nuova eccezione che permette di segnalare al client che qualcosa non va nel file di input (in questo caso si tratta sempre di errori di id) qualora questo problema fosse trovato in fase di risoluzione (cioè eseguendo il metodo *solve*).

2 - Robustezza

La robustezza dell'applicazione è fornita gestendo opportunamente le eccezioni che i diversi metodi remoti possono sollevare:

2.1 - Lato client

- **NotBoundException**: viene sollevata quando il nome cercato dal client non sia stato caricato sul registro, in questo caso viene stampato a video il messaggio "Errore: il nome cercato non è stato caricato nel registro."

- **RemoteException**: potrebbe venire sollevata in diversi casi (nota queste eccezioni vengono sollevate anche quando la comunicazione col server si interrompe improvvisamente)
    - *Lookup dell'oggetto server*: indica che c'è stato un errore (diverso da quello precedente) nel lookup dell'oggetto server, viene stampato a video il messaggio "Errore: c'è stato un errore nel lookup del riferimento remoto."
    - *server.getPuzzle()*: c'è stato un errore nell'invocazione del metodo remoto *getPuzzle*, viene stampato a video "Errore: non è stato possibile ottenere un riferimento a un puzzle remoto."
    - *puzzle.solve()*: c'è stato un errore nell'invocazione del metodo remoto *solve()*, viene stampato a video il messaggio "Errore: c'è stato qualche errore di connessione nella risoluzione remota del puzzle."

2.2 - Lato server

- **MalformedURLException**: stampa a video "Errore: il nome fornito sembra dare problemi."
- **RemoteException**:
    - *LocateRegistry.createRegistry(2020)*: c'è stato un errore nella fase di creazione del registro, stampa a video "Errore: c'è stato un errore nella creazione del registro."
    - *Naming.rebind*: c'è stato un errore nel rebind dell'oggetto server, viene stampato a video "Errore: c'è stato un errore nel contattare il registro."

2.3 - File di input

Vengono eseguiti gli stessi controlli della parte precedente sul file di input. Gli errori possono essere trovati all'interno dei due metodi dell'interfaccia *Puzzle* (implementati in *ConcurrentPuzzleImpl*)

- *initialize*: qualora in questo metodo fosse individuato un problema nel file di input verrà ritornata al chiamante una stringa contenente la descrizione di tale errore, una stringa vuota indica la corretta inizializzazione del puzzle; il client copierà la stringa di errore nel file di output e stamperà a video un messaggio di errore invitando a guardarvi dentro per maggiori informazioni
- *solve*: qualora in questo metodo fosse individuato un problema (a questo punto si può trattare solo di mismatch di id), una stringa esplicativa viene costruita (una per ogni errore individuato) e al termine del metodo viene sollevata un'eccezione *WrongInput* che viene gestita dal client stampando a video un messaggio di errore ed invitando a guardare nel file di output per ottenere maggiori informazioni

3 - Logica client / server

La comunicazione tra client e server avviene seguendo questi passi

1. Il client legge il file di input e lo copia dentro una stringa
2. Il client ottiene un riferimento remoto all'oggetto server e, a partire da questo, un riferimento remoto a un oggetto puzzle
3. Il client inizializza il puzzle remoto con la stringa di input precedentemente ottenuta
4. Il client invoca il metodo *solve* del puzzle remoto e ottiene una stringa contenente il risultato