**Università degli Studi di Bologna**
**Scuola di Ingegneria**

# Corso di
# Reti di Calcolatori T

***Esercitazione 6 (proposta)***
***Java RMI***

**Luca Foschini**

Anno accademico 2013/2014



## Specifica

Utilizzando java RMI sviluppare **un'applicazione C/S** che consenta di effettuare le operazioni remote per:

- **contare i caratteri**, **le parole** (si usi la definizione di parola data nell'esercitazione 1) e **le linee** di un **file di testo** presente sul server remoto;
- **restituire la lista con i nomi di file** (presenti nel **direttorio remoto** indicato dal client) **il cui nome finisce col suffisso richiesto dal client** (ad esempio ".txt").

I clienti siano sempre ciclici e condizionati a lavorare fino ad esaurire tutti gli input dall'utente; il server deve comportarsi da processo sempre presente.

# Metodi remoti

Il progetto RMI si basa su un'**interfaccia remotizzabile** (**RemOp,** contenuta nel file RemOp.java) in cui vengono definiti i **metodi invocabili in remoto dal client**:

- Il metodo `conta_fileTesto` accetta come parametro d'ingresso **il nome del file**, quindi restituisce un array di **tre interi** corrispondenti al numero di occorrenze di caratteri, parole e linee contate; altrimenti, in caso di errore, solleva un'*eccezione remota*: ad esempio, se il file non è presente nel sistema.
- Il metodo `lista_file_suff` accetta come parametri d'ingresso **il nome del direttorio e il suffisso**; quindi, se il direttorio richiesto esiste, lo apre e restituisce al client la **lista dei file** (come array di stringhe) il cui nome termina col suffisso richiesto; in caso di errore, solleva un'eccezione remota: ad esempio, se il direttorio non esiste.



# Classi in gioco

Si progettino le classi:

- **ServerImpl** (contenuta nel file ServerImpl.java), che implementa i metodi del server invocabili in remoto e presenta l'interfaccia di invocazione:

  **ServerImpl [registryPort]**

- **Client** (contenuta nel file Client.java), che realizza l'interazione con l'utente proponendo ciclicamente i servizi che utilizzano i due metodi remoti, e stampa a video i risultati, fino alla fine del file di input da tastiera. Il Client presenta l'interfaccia di invocazione:

  **Client NomeHost [registryPort]**

# Note per recupero riferimento Server

Il **Registry** deve essere in esecuzione su un host concordato e in ascolto alla porta eventualmente specificata.

Il **Server** (istanza della classe relativa) deve registrare il riferimento remoto sul registry alla locazione corretta.

Il **Client** (istanza della classe relativa) deve recuperare dal registry il riferimento all'oggetto remoto, ServerImpl, di cui deve invocare i metodi.



# Proposta di estensione: Trasferimento di un direttorio

Si vuole sviluppare **un'applicazione C/S basata su RMI** e su socket con connessione per il trasferimento di tutti i file di un direttorio remoto dal server al client (**multiple get**). In particolare, si vogliono realizzare due modalità di trasferimento, la prima con **client attivo** (**il client effettua la connect**), la seconda con **server attivo** (**il server effettua la connect**). Si dovranno realizzare un **client** e un **server**; l'utente, per ogni trasferimento, decide quale delle due modalità utilizzare.

Per entrambe le modalità, si prevede **un'interazione iniziale sincrona (realizzata con una richiesta RMI sull'oggetto remoto server)** per trasferire la **lista dei file** da inviare e **l'endpoint** (host e porta) **di ascolto**; quindi, una seconda fase di **trasferimento dei file** dal server al client (realizzata con socket connesse).

# Trasferimento più direttori: Client Attivo

Il metodo remoto accetta come argomento di ingresso il **nome del direttorio** e restituisce una struttura dati con l'**endpoint di ascolto del server** e la **lista con i nomi e la lunghezza di tutti i file** da trasferire.

Il **Client** richiede ciclicamente all'utente **il nome del direttorio** da trasferire ed **effettua la chiamata RMI e riceve l'enpoint di ascolto**, quindi **stabilisce una connessione** con il server remoto e riceve i file salvandoli sul direttorio locale.

Il **Server** implementa **il metodo RMI richiesto** ed è realizzato come **server concorrente** e **parallelo**. Per ogni nuova richiesta ricevuta il processo padre, dopo **aver accettato la richiesta RMI**, attiva un processo figlio a cui affida la **creazione della socket di ascolto** e il completamento del servizio richiesto; quindi il padre restituisce **la lista dei file** e **il proprio endpoint**.



# Trasferimento più direttori: Server Attivo

Il metodo remoto accetta come argomento di ingresso il **nome del direttorio** e l'**endpoint di ascolto del client** e restituisce la **lista con i nomi e la lunghezza di tutti i file** da trasferire.

Il **Client** richiede ciclicamente all'utente **il nome del direttorio** da trasferire, **crea la socket di ascolto**, **poi effettua la chiamata RMI** e riceve la lista dei file da trasferire; infine, **effettua la accept** e i trasferimenti di file necessari.

Il **Server** implementa il metodo RMI richiesto ed è realizzato come **server concorrente** e **parallelo**; per ogni nuova richiesta ricevuta il processo padre, dopo aver **accettato la richiesta RMI**, attiva un processo figlio a cui affida la **creazione della socket** su cui eseguire la **connect** e il completamento del servizio richiesto; quindi **restituisce la lista dei file**.

# Consegna

*Chi vuole può inviare via email lo svolgimento dell'estensione ai docenti, in modo da discutere la soluzione ed eventualmente pubblicarla sul sito del corso*