Progetto 1 Sistemi Distribuiti 2021-2022

Descrizione del sistema da realizzare
(gruppi da max 2 persone, consegna 23 Giugno entro le 24:00 su sito elearning)

Si desidera realizzare il sistema distribuito costituito da vari server e client. I server e i client possono risiedere su macchine diverse, come in ogni sistema distribuito: **il codice quindi va preparato nell'ottica di una distribuzione vera e propria, anche se poi i test verranno eseguiti in locale**.

Si desidera realizzare un meccanismo che permetta a dei server che offrono dei servizi (identificati da un nome) di registrare quei servizi presso un *Registry* situato su una macchina nota. I client possono accedere al Registry, cercare il server che offre un certo servizio, recuperarne le informazioni per contattarlo, e quindi aprire un canale con quel server per richiedere il servizio.
Tutte le comunicazioni devono essere implementate utilizzando socket o channels (potete usare readline e writeline per i socket e i channels).

Il sistema deve prevedere quindi le seguenti entità: un server *Registry*, n *Server* (in grado di offrire i servizi richiesti nel testo più avanti), m *Client* che desiderino utilizzare quei servizi.

Step di registrazione e ricerca servizi:

- Il Registry (in esecuzione su una macchina nota ed in ascolto su una porta nota) deve offrire le seguenti operazioni: *bind* (registrazione di un nuovo servizio da parte di un server), *unbind* (cancellazione di un servizio offerto da un dato server), *rebind* (che sovrascrive, per un dato servizio, le informazioni del server che lo offre), e *lookup* (ricerca di un dato servizio tramite il nome identificativo).
- Un server pubblica il nome del servizio offerto, il proprio riferimento remoto (costituito da indirizzo IP e porta) e il nome del protocollo applicativo sul Registry mediante una operazione di *Boolean bind(String nomeServizio, String IP server, int Porta, String Protocollo);*
    - NB: per semplificare, assumiamo che un servizio possa essere offerto da uno e un solo server. Quindi, sul Registry i servizi sono univocamente identificati dal loro nome
    - Un server può essere in grado di offrire servizi differenti (si veda esempio successivo)
- il client reperisce il riferimento remoto al server (IP, porta e protocollo applicativo) cercandolo sul Registry, con una operazione di *String lookup(nomeServizio);*
    - l'indirizzo della macchina su cui è in esecuzione il Registry deve essere quindi passato ai client e ai server come parametro (mentre la porta può essere fissa e scelta da voi a priori)
- una volta recuperato l'indirizzo del server che offre il servizio desiderato, il client lo contatta seguendo il protocollo applicativo specificato. Assumiamo che i vari protocolli applicativi di interazione client – server associati ai servizi siano noti (si veda esempio successivo)
- **NB: Il Registry può essere implementato in maniera semplice, gestendo le richieste dei client anche in maniera sequenziale (senza quindi necessariamente channels o thread)**

Implementate un Server che offra il servizio bancario “contoCorrenteMiaBanca” come descritto di seguito:

- Il server mantiene un elenco di *contiCorrenti*, identificati da un numero univoco (identificatore assegnato dal server in fase di creazione nuovo conto) e dal codice cliente (anche questo univoco). Un cliente può avere più conti, un conto più clienti.
- Il server che gestisce il servizio bancario accetta le seguenti operazioni:
    - *Integer nuovoConto([codiceCliente])* che prende in input **una lista di nomi clienti** (gli intestatari del conto) e restituisce un nuovo numero di conto
    - *Boolean versa(idConto, codiceCliente, quantita)* che versa sul conto una certa cifra (se il conto esiste ed è intestato [anche] a quella persona, altrimenti ritorna false)
    - *Boolean preleva(idConto, codiceCliente, quantita)* che preleva dal conto una certa cifra (se il conto esiste ed è intestato [anche] a quella persona, e quella cifra è disponibile, altrimenti ritorna false)
    - *Boolean bonifico(idContoPrelievo, codiceClientePrelievo, idContoVersamento, quantita),* che preleva *quantita* dal conto *idContoPrelievo* associato al *codiceClientePrelievo* e la versa su *idContoVersamento*. L’operazione va a buon fine se entrambi i conti esistono, se il primo è affettivamente intestato [anche] a *codiceClientePrelievo*, e se su *idContoPrelievo* è disponibile *quantita*.
- **Il server deve creare un thread slave dedicato per ogni nuova richiesta da parte di un client: lo slave gestisce una sola richiesta di servizio e poi termina**
- **Il server deve garantire che l’accesso ai conti correnti sia thread safe, e NON deve utilizzare strutture Java già thread-safe (concurrent hashmaps, blocking queue etc)**

Il client che desidera accedere al servizio “contoCorrenteMiaBanca” come sopra descritto **deve seguire i seguenti step**:

- Recupera, presso il Registry, l’indirizzo del server che offre il servizio “contoCorrenteMiaBanca”
- Crea un nuovo conto corrente, eventualmente cointestato con un altro client (il cui codice cliente potete decidere voi come sceglierlo, se random, leggendolo da input, da utente, o da file etc)
- Crea una struttura di dimensione fissa da voi scelta (si suggerisce 2 o 3) che tenga l’elenco delle operazioni (create dinamicamente) che vuole svolgere sul conto corrente
- Crea un thread che, con frequenza random, crea una nuova operazione random (versamento, prelievo, bonifico) sul conto e tenta di inserirla nella struttura condivisa, se c’è posto (altrimenti attende)
- Il thread main, quando trova nella struttura condivisa una nuova operazione, la comunica al server, stampa a video il risultato, e si rimette in attesa di una nuova operazione da eseguire. Dopo un tot di secondi, termina tutto
- **NB: anche in questo caso, non potete usare strutture Java già thread safe.**
- **NB2: scegliete voi come selezionare uno, o più, conti correnti su cui fare il bonifico (random, aggiunta di un metodo sul server che vi dà l’elenco dei conti presenti (sorvoliamo sui problemi di privacy), richiesta all’utente, etc.)**

Quanto non specificato riguardo ai protocolli applicativi è lasciato a vostra scelta (ad esempio, se usare una writeline o mandare una stringa unica, che carattere separatore eventualmente usare, i tempi random, le eventuali sleep etc). In caso di dubbi, chiedete sul forum (dedicato)!

Per testare il sistema, si può implementare ad esempio:

1. Un processo Registry
2. Un server che registra il suo servizio di "contoCorrenteMiaBanca" sul Registry, e poi aspetta chiamate dai client
3. Due client che si rivolgono al Registry per recuperare il riferimento remoto al server per il servizio di "contoCorrenteMiaBanca", e poi generano casualmente diverse richieste di operazioni, che vengono inviate al server. Chiaramente, se riusciste a creare conti cointestati ai due client, e conti differenti su cui fare i bonifici, potreste verificare meglio eventuali problemi di concorrenza…
4. Per testare meglio il registry, potete ad esempio aggiungere un server (e un client) che riusino uno dei servizi implementati a lezione/esercitazione, tipo il protocollo di echo (questo è opzionale)

Una configurazione di esempio del Registry potrebbe essere la seguente:

contoCorrenteMiaBanca: <IP1,porta1,ProtocolloMiaBanca>

servizioX: <IP1, porta3, ProtocolloX> (ok l'utilizzo dello stesso IP ma porte diverse e servizi diversi)

servizioY: <IP2, porta2, ProtocolloY>

echo: <IP2, porta3, ProtocolloEcho>

contoCorrenteMiaBanca: <IP2,porta2,ProtocolloMiaBanca> NO (servizio duplicato)!!

(IP1, IP2 etc. possono essere tutti localhost)

Implementare i programmi in modo che scrivano a terminale cosa stanno facendo.
Ogni volta che rilanciate Registry, si riparte da zero (non c'è bisogno di tenere memoria dei servizi o di altro).
Non complicate la soluzione inutilmente, né a livello di progettazione generale né di algoritmi…

Fornire anche un file *readme* che descriva la soluzione ad alto livello, e che presenti le istruzioni per il lancio del sistema (includendo quindi istruzioni di compilazione, sequenza dei processi che devono essere lanciati, eventuali parametri, etc.).
Potete creare degli script per lanciare i vari client e server (specificando per quale SO sono valide), o almeno dovete fornire la sequenza precisa, e le istruzioni, per lanciare i vari processi. Anche test JUnit (v4) sono estremamente utili per facilitare il processo di correzione.

Il progetto deve essere scritto in Java, eseguibile o come progetto Eclipse o alla peggio compilabile (seguendo le istruzioni da voi date) ed eseguibile da shell. Progetti che richiedano altri IDE, o interventi manuali per essere resi eseguibili, o che non compilano, non verranno valutati.
Consegnate uno zip direttamente sul sito di elearning, nella consegna relativa a questo progetto.
Lo zip e il progetto devono riportare i cognomi dei componenti del gruppo nel seguente formato: MATRICOLA1_Nome1_Cognome1_MATRICOLA2_Nome2_Cognome2.zip. Se uno studente svolge il progetto individualmente, il nome del file deve essere nel formato MATRICOLA_Nome_Cognome.zip.