Relazione di Stefano Carbone 654222 per il progetto di lab3

Comandi

- java -jar Client.jar -> avvia il client
- java -jar Server.jar -> avvia il server
- Le varie opzioni dell'interfaccia del client sono selezionabili tramite il tastierino numerico
- Per scegliere il tipo di ordine scrivere : ask o bid
- Ctrl+c -> Termina il server

Classi del Progetto:

Server

- legge il file di configurazione
- carica la Hashmap dal file Json (se non esiste lo crea)
- carica l'orderbook con tutti gli ordini del file Json (se non esiste lo crea)
- crea cachedThreadePool, scelgo questo perché si può espandere all'infinito
- crea socket che sta in attesa delle connessioni TCP
- crea socket per le connessioni UDP
- crea un Thread (TerminationHandler) che si occupa della chiusura del server in modo safe tramite il comando ctrl+c
- crea un Thread che esegue gli stop order
- Tramite un ciclo sta in attesa di nuove connessioni TCP
    - Crea Thread nel pool che gestisce la nuova connessione
- Nella sua classe sono contenute le funzioni sendNotification per inviare le notifiche di invio pacchetto, la funzione di lettura dal file di configurazione e la funzione dello stopOrderThread.

StopOrderThread: Si occupa di eseguire gli ordini stop quando il prezzo e la quantità sono corrette

Client

- Fornisce all'utente una sezione di comandi per interagire con il server
- Ogni funzionalità permette di creare una richiesta in formato json che verrà inviata al server
- Le risposte vengono poi deserializzate e riportate a schermo
- Tramite due funzioni receiveReply e sendRequest posso ricevere ed inviare messaggi in formato json
- Crea un thread che riceve le notifiche UDP dopo l'avvenuta registrazione
- La funzione che gestisce il Thread si chiama UDP_reciver ed è implementata in questa classe, alla ricezione di un datagramma dal server, estrae il messaggio, lo deserializza e stampa il vettore di ordini eseguiti a schermo.

Worker

- La classe worker implementa il Thread per gestire le nuove connessioni TCP
- Crea due variabili per leggere e scrivere nello stream
- Sta in attesa delle richieste da parte del client
- Ricevuta una richiesta la deserializza e la filtra in base al tipo di operazione
- Contiene una funzione sendReply per mandare le risposte serializzate al client

OrderBook

- L'orderBook implementa le varie strutture e le funzioni per gestire gli ordini
- Contiene una variabile atomica orderID che conta il numero di ordini
- Gli ordini limit sono contenuti all'interno di due PriorityBlockingQueue, una per gli ordini buy e una per gli ordini sell ordinati in base ad un comparatore
- Gli ordini stop sono contenuti all'interno di una ConcurrentHashMap e vengono controllati ed eseguiti dal thread che si trova nel main
- Le strutture sono implementate Singleton in modo che non possono essere duplicate
- L'orderBook contiene le funzioni per eseguire gli ordini in modo parziale o totale tramite la funzione executOrder.
- Tutti gli ordini evasi vengono salvati in un file json in automatico tramite la funzione saveExecuteOrder (che viene creato in automatico se non presente)
- Alla fine della classe troviamo la funzione findUserFromId che viene utilizzata per trovare gli utenti in modo tale da inviare le notifiche dopo l'esecuzione di un file

CHashMap

- La CHashMap contiene l'implementazione della struttura per conservare gli utenti
- Viene usata una ConcurrentHashMap implementata Singleton
- La classe contiene le varie funzioni che permettono di inserire, modificare, ecc la mappa

TerminationHandler

- Questa classe descrive il funzionamento del Thread TerminationHandler
- Il thread si occupa della chiusura safe del server
- In particolare chiude il pool di thread , salva gli account delle mappa nel file Json corrispondente , salva gli ordini delle code/mappe nel file Json corrispondente ed infine chiude le connessioni

Classi che implementano gli oggetti:

- Order -> Implementa l'ordine e le funzioni per gestirlo
- Account -> Implementa l'account e le funzioni per gestirlo
- Reply -> Implementa il messaggio di risposta
- Operation-> Implementa il messaggio di richiesta
- Date -> Implementa il tipo della richiesta "getPriceHistory"
- PriceStatistic -> Implementa il messaggio di risposta per "getPriceHisotry" usato dal worker

SERIALIZZAZIONE/DESERIALIZZAZIONE

Principalmente avvengono tutte usando la funzione toJson/fromJson di GSON

ALTRI FILE:

Il Programma fa riferimento a diversi file tra cui:

- ClientConfig -> file di configurazione del client
- ServerConfig -> file di configurazione del server
- ordiniEvasi -> file json che contiene gli ordini evasi
- storicoOrdini -> file json che contiene gli ordini da inserire in memoria
- utenti -> file json che contiene gli utenti da inserire in memoria