RELAZIONE

checkWhichTransactionValid()

Il metodo checkWhichTransactionValid() serve a controllare se le stringhe presenti in mRequest sono valide o meno. Per poter procedere alla validazione è necessaria la connessione al server, il quale dispone di tutti i rami che compongono il merkle tree.

Il metodo comincia dichiarando e inizializzando delle variabili d'ausilio. In seguito, si entra dentro un blocco try catch al fine di gestire le eventuali eccezioni che possono sorgere dal tentativo di connettersi al server o dalle operazioni di input e output. La connessione dura fino a quando il server ha fornito le informazioni necessarie poichè, personalmente, ritengo inutile mantenere la connessione per l'intera durata del main. Per poter sapere quando uscire dalla connessione ho optato per la creazione di una nuova stringa da aggiungere alla mRequest: "close".

Una volta che la connessione è avvenuta con successo e si è aggiornata la mRequest si entra in un ciclo for, nel quale si processano tutte le stringhe presenti in mRequest. Il buffer di riferimento viene allocato con una dimensione stabilita dal server cosa resa possibile dal metodo socket() seguito da getReceiveBufferSize(). Il motivo di questa implementazione è legato al fatto che il server conosce la dimensione del merkle tree e sa quanto buffer è necessario allocare considerando il caso peggiore; quindi si procede con la scrittura nel canale dei dati e lettura del risultato ottenuto dal server. Tale risultato viene controllato dalla isTransactionValid(String merkleTx, List<String> merkleNodes) e si procede all'inserimento del messaggio corrente nella corretta lista temporanea.

Al termine della connessione ho optato per rimuovere dalla mRequest il valore "close" in quanto è solo una stringa temporaneo e quindi chiudo la connessione e torno in locale. Completo il metodo inserendo le due liste temporanee nella Map<boolean, List<String>> che verrà restituita.

isTransactionValid(String merkleTx, List<String> merkleNodes)

Il metodo in questione ottiene in merkleTx una stringa di mRequest che in quel momento deve essere validata, mentre in merkleNodes gli viene passata la lista di stringhe fornite dal server nell'ordine corretto in cui devono essere eseguite. Sia i dati di mRequest sia quelli passati dal server sono già stati codificati in md5Java quindi resta solo la necessità di concatenare le stringhe ottenute e di rieseguire md5Java sul risultato. Infine, bisogna verificare che il risultato ottenuto una volta risalito l'albero coincida con mRoot.

SERVER

Sebbene la consegna non alluda a richieste di implementazione del server ho ritenuto necessario crearne uno per effettuare un test sul corretto funzionamento dell'esercizio. In relazione a questo, desidero evidenziare una riga di codice ovvero la numero 34:

```
serverSocket.socket().setReceiveBufferSize(256);
```

quest'ultima è stata inserita per settare la dimensione del buffer in modo che il client possa accedere al server e sapere quanto deve allocare nel buffer.

SCELTA

Per l'esercizio assegnato ho ritenuto superficiale l'uso dei thread in quanto l'applicazione non necessita di eseguire parallelamente operazioni o di condividere dati.

L'unica operazione che utilizza la classe Thread è la sleep() in riga 117 questo al fine verificare la seguente situazione nella parte server:

```
i'm a server and i'm waiting for new connection and buffer select...
--- Message received: 0ff89de99d4a8f4b04cb162bcb5740cf
i'm a server and i'm waiting for new connection and buffer select...
--- Message received: 8ca10608a248910c25083c3dab4371c3
```