Università di Catania

DIEEI

Dipartimento di Ingegneria Elettrica Elettronica e Informatica

Ingegneria Informatica Magistrale LM-32

Corso di Internet of Things based Smart Systems a.a. 2021/2022

Vincenzo Pluchino 1000023070

Relazione elaborato: interfaccia grafica Java per il progetto Bitcoffee

Insegnante: prof. Davide Patti

Sommario

1 Introduzione

Il progetto consiste nella creazione di un'interfaccia grafica per il progetto (ancora in corso di sviluppo) creato dal professore Patti chiamato "bitcoffee".
L'obiettivo di bitcoffee è quello di reimplementare da zero (in linguaggio Java) tutte le funzionalità di un nodo Bitcoin, basandosi sul libro di Jimmy Song "Programming Bitcoin" di cui è possibile consultare molte informazioni sul sito web https://programmingbitcoin.com/

Nel progetto sono presenti diverse funzioni e caratteristiche utilizzabili, tra cui:

- Algoritmo per firma digitale basato sulle curve ellittiche (secp256k1), aritmetica dei campi finiti, little/big endian and hex utilities e altre funzionalità simili (SEC, DER, calcolo hash SHA256, RIPEMD160, decodifica Base58)
- Serializzazione e analisi delle transazioni di Bitcoin
- Linguaggio di Script Bitcoin per esecuzione e analisi
- Analisi della difficoltà dei blocchi e proof-of-work
- Possibilità di connettersi ai nodi esterni per recuperare e validare i blocchi
- Algoritmi di Segregated Witness per il cambio soft fork atto a risolvere i problemi di scalabilità presenti nel formato delle transazioni di Bitcoin

2 Funzionalità implementate nell'interfaccia

In totale i "servizi" da dover far eseguire all'interno dell'interfaccia grafica sono otto. Di seguito verranno introdotti singolarmente, fornendo una breve descrizione di come si utilizzano e per quali scopi sono stati sviluppati.

1) Signature

Questa funzione consente di creare una firma digitale utilizzando due stringhe di cui una costituirà il messaggio e l'altra un "secret", ovvero una sorta di password univoca che consente di verificare se la firma è autentica o meno. Può anche essere utilizzata per creare una chiave privata.

```
private static void cmd_sign(String secret, String message) {
    // input: any two strings are valid
    var secret_bytes = Kit.hash256(secret);
    var secret_num = new BigInteger(1, secret_bytes);
    var msg_bytes = Kit.hash256(message);
    var msg_num = new BigInteger(1, msg_bytes);
    System.out.println("Signing (secret:" + secret + " message:" + message + ")");

    System.out.println("secret hash: " + secret_num.toString(16));
    System.out.println("msg    hash: " + msg_num.toString(16));
    var pk = new PrivateKey(secret_bytes);
    var signature = pk.signDeterminisk(msg_bytes);
    System.out.println("signature: " + signature.toString());
}
```

Da notare l'uso degli hash su entrambe le stringhe che consentono di avere una certa sicurezza sull'autenticità della firma

2) Parse block

La funzione di parse block ha l'obiettivo di analizzare il blocco inserito come input, andando a leggere la block chain di Bitcoin e visualizzare dove sia inserita al suo interno. Fornisce anche ulteriori informazioni, come il BIP (Bitcoin Improvement Proposal), l'hash del blocco e la sua difficoltà.

```
private static void cmd_parseblock(String block_raw) {
    // input: a string representing a block serialization in hex
    //block_raw = "020000208ec39428b17323fa0ddec8e887b4a7c53b8c0a0a220cf
    System.out.println("Parsing raw block: " + block_raw);

    var block = Block.parseSerial(Kit.hexStringToByteArray(block_raw));
    System.out.println(block);
    System.out.println("-----------------------------------------------
    System.out.println(" details");
    System.out.println("-----------------------------------------------
    System.out.print("BIP: ");
    assert block != null;
    if (block.checkBIP9()) System.out.println("BIP9");
    if (block.checkBIP91()) System.out.println("BIP91");
    if (block.checkBIP141()) System.out.println("BIP91");
    System.out.println("block hash: " + block.getHashHexString());
    System.out.println("difficulty: " + block.difficulty());

    System.out.println();
}
```

L'input deve essere la serializzazione del blocco in esadecimale

3) Difficulty adjustment

L'operazione di difficulty adjustment consente di calcolare la differenza di difficoltà esistente tra un blocco iniziale e un blocco finale, visualizzando la

differenza di tempo impiegata per trovare i due blocchi e suggerendo quali bit dovrebbero essere impiegati per aggiustare e quindi mantenere costante la difficoltà nella ricerca dei blocchi.

```
    var first_block = Block.parseSerial(Kit.hexStringToByteArray(start_block));
    var last_block = Block.parseSerial(Kit.hexStringToByteArray(end_block));
    System.out.println("Computing difficulty adjustment...");

    System.out.println("First block:");
    System.out.println("-----------------------------------------------");
    System.out.println(first_block);
    System.out.println("Last block:");
    System.out.println("-----------------------------------------------");
    System.out.println(last_block);
    assert last_block != null;
    assert first_block != null;
    var time_diff = last_block.getTimestamp() - first_block.getTimestamp();
    System.out.println("Time differential: " + time_diff + ", updating bits: " + Kit.bytesToHexString(first_block.getBits()));

    var new_bits = Block.computeNewBits(first_block.getBits(), time_diff);
    System.out.println("New bits: " + Kit.bytesToHexString(new_bits));

}
```

4) Getp2pkaddr

Bitcoffee tramite questo comando offre un tool per ricavare una propria chiave privata tramite una stringa secret, per poi utilizzarla in alcuni indirizzi come, ad esempio, il proprio wallet Bitcoin.
Getp2pkaddr, insieme ad altre funzioni, può essere utilizzate su due reti: testnet e mainnet. La differenza tra le due è molto importante: Mainnet (abbreviazione di main network) è la blockchain originale e funzionale in cui le transazioni effettive avvengono nel registro distribuito e la criptovaluta nativa possiede un reale valore economico. In altre parole, la mainnet si riferisce alla stessa blockchain open source che è verificabile pubblicamente.
Testnet (abbreviazione di test network) è una blockchain alternativa, da utilizzare per i test. I token su Testnet sono separati e distinti dai token effettivi della blockchain originale e non dovrebbero mai avere alcun valore. Ciò consente agli sviluppatori di applicazioni o ai tester di sperimentare senza dover utilizzare un Bitcoin reale o preoccuparsi di interrompere la blockchain

principale.

```
private static void cmd_getp2pkaddr(String secret, boolean testnet) {
    // input: a secret string
    // brainwallet style, use string text to derive private key (be careful to not share
    var secret_bytes = Kit.hash256(secret);
    var mypk = new PrivateKey(secret_bytes);

    var myaddress = mypk.point.getP2pkhTestnetAddress();
    if (testnet)
        System.out.println("Testnet address for secret:" + secret);
    else
        System.out.println("Mainnet address for secret:" + secret);

    System.out.println("address: " + myaddress);
    var wif = mypk.getWIF(true, true);
    System.out.println("Use this WIF to import the private key into a wallet: " + wif);
    System.out.println("--------------------------------------------");

    System.out.println("--------------------------------------------");
}
```

Il WIF (Wallet Import Format) è un modo per codificare una chiave ECDSA privata in modo da semplificarne la copia

5) Check transaction

L'operazione di check è utilizzata per la verifica di una transazione; l'input da immettere è l'identificativo della transazione serializzata rappresentato in esadecimale. Il controllo avviene principalmente sulle fees (commissioni) della transazione, per vedere se esse siano valide oppure no.
In generale, le fees di una transazione sono la differenza tra l'importo di bitcoin inviato e l'importo ricevuto: in poche parole riflettono la velocità con cui un utente desidera che la propria transazione venga convalidata sulla blockchain. Quando un miner convalida un nuovo blocco nella blockchain, convalida anche tutte le transazioni all'interno del blocco.

```
private static void cmd_checkTx(String raw_tx) {
    // input: a string representing the hex of a serialized tx
    // for examples, fetch from online services e copy the resulting content:
    // https://blockstream.info/api/tx/716373514d1442f6e7f71719965936fc8df12fe

    System.out.println("-------------------------------------------------");
    var tx = Tx.parse(Kit.hexStringToByteArray(raw_tx), false);
    System.out.println(">> Checking fee for tx id:" + tx.getId());
    var fee = tx.calculateFee();
    if (fee >= 0)
        System.out.println("Valid transactions fees:" + fee);
    else
        System.out.println("ERROR: not valid transaction fees:" + fee);
    System.out.println("-------------------------------------------------");
}
```

6) Verify signature

Un altro tool interessante è quello della verifica della validità di una firma, effettuata tramite alcuni input in esadecimale come un numero intero (z), stringa crittografata con crittografia a curva ellittica (SEC, Standards for Efficient Cryptography) e un altro parametro.

```
private static void cmd_verifySignature(String sec, String der, String z) {
    // inputs: hex strings representing sec,der,z
    //sec = "0349fc4e631e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278a";
    //der = "3045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f
    //z = "27e0c5994dec7824e56dec6b2fcb342eb7cdb0d0957c2fce9882f715e85d81a6";

    var z_num = new BigInteger(z, 16);
    var point = S256Point.parseSEC(sec);
    var signature = Signature.parse(Kit.hexStringToByteArray(der));
    System.out.println(" >> Verify signature test: " + point.verify(z_num, signature));
```

7) Get transaction

La funzione “get transaction” ha la missione di ottenere la transazione, permettendo di visualizzare in quale indirizzo della rete si trova e quale sia il suo id. Per potere utilizzarla è necessario fornire l’ultimo blocco della transazione, l’host e l’indirizzo in cui dovrebbe trovarsi. Anche qui, come nella funzione p2pkaddr descritta al punto 4, è possibile eseguire questo strumento sia su testnet che su mainnet.

```
while (!found) {

    var msg = node.waitFor(msg_to_wait);

    if (msg != null) {

        if (msg.getCommand().equals("merkleblock")) {

            if (!((MerkleBlock) msg).isValid())
                throw new RuntimeException("Not valid Merkle proof");
            else System.out.println("Received valid Merkle block");
        } else {
            var receveived_tx = (Tx) msg;
            for (TxOut tout : receveived_tx.getTxOuts()) {
                if (tout.getScriptPubKey().getAddress(testnet).equals(address)) {
                    System.out.println("Found address " + address + " in tx id: " + receveived_tx.getId());
                    found = true;
                }
            }
```

Queste sono solamente alcune righe di codice della funzione "get transaction". È una delle più complesse di bitcoffee.

8) Fetch transaction

Sebbene possa sembrare simile al comando di check descritto nel punto 5, l’istruzione di fetch (ovvero di recupero) si pone l’obiettivo di verificare la validità (e conseguentemente l’esistenza) di una transazione in una delle due reti (test o main). Per effettuare questa verifica è necessario fornire l’id della

transazione e selezionare la rete testnet oppure mainnet.

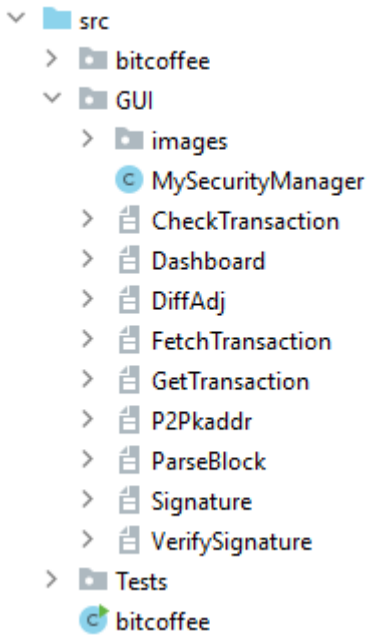
```java
private static void cmd_fetchtx(String txid, boolean testnet) {
    // inputs: the transaction id and the whether is testnet or mainnet
    // look at TestSegwit.java for example inputs
    var tx = TxFetcher.fetch(txid,testnet,true);
    System.out.println(tx);

    System.out.println("-----------------------------------------");
    System.out.println("Verifying transaction:");
    System.out.println("-----------------------------------------");
    if (tx.verify()) System.out.println("--> Transaction confirmed as valid");
    else System.out.println("--> Transaction is NOT valid!");

}
```

3 Creazione della GUI (Graphical User Interface) di bitcoffee

Per lo sviluppo dell'interfaccia grafica è stato utilizzato l'IDE IntelliJ, che dispone al suo interno uno dei framework più utilizzati per la creazione di UI (User Interface) in Java: Swing.



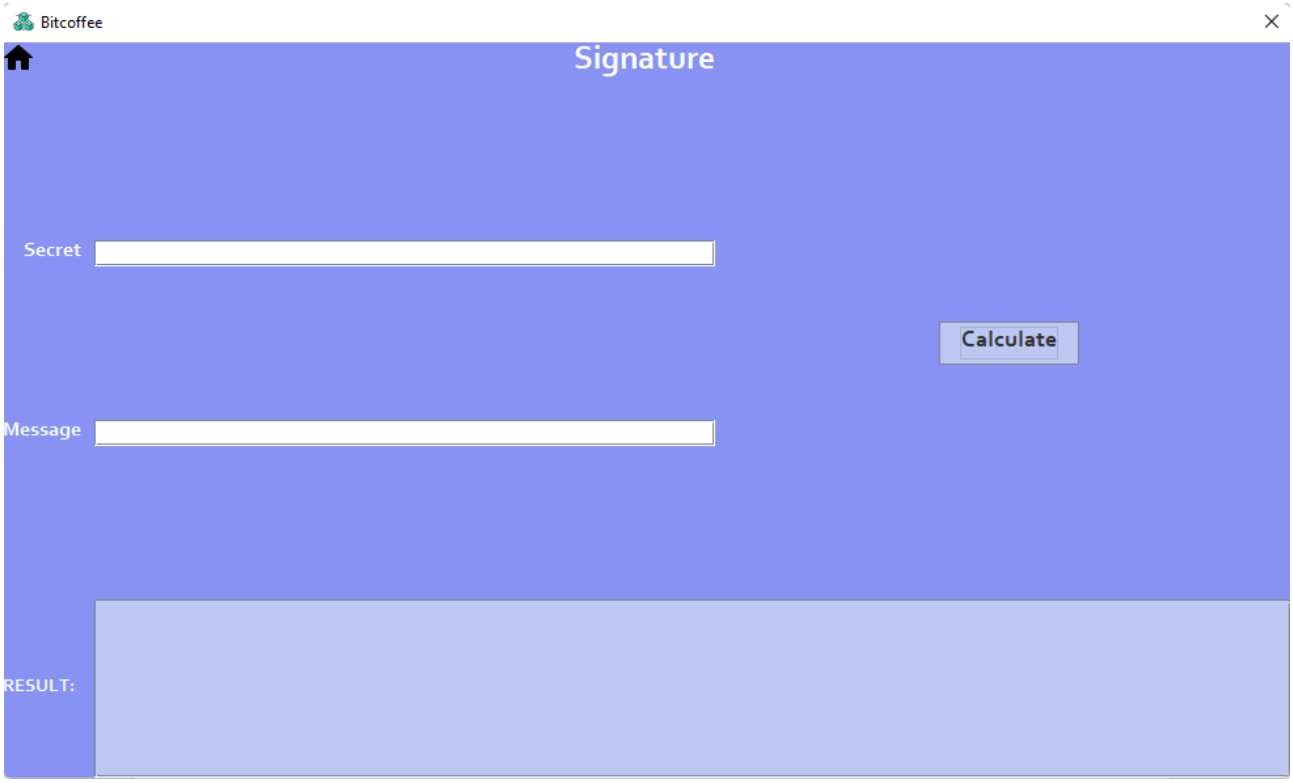
Organizzazione dei packages e dei file di bitcoffee

Come è possibile notare dall'organizzazione dei file, tutti i frame e le classi relative alla parte grafica sono contenuti all'interno del package "GUI"; questo consente di lavorare in modo indipendente con gli altri packages e quindi di continuare a sviluppare nuove funzionalità del progetto senza che l'interfaccia vada ad interferire con le classi di funzionamento principali (sono quelle contenute nel package "bitcoffee").

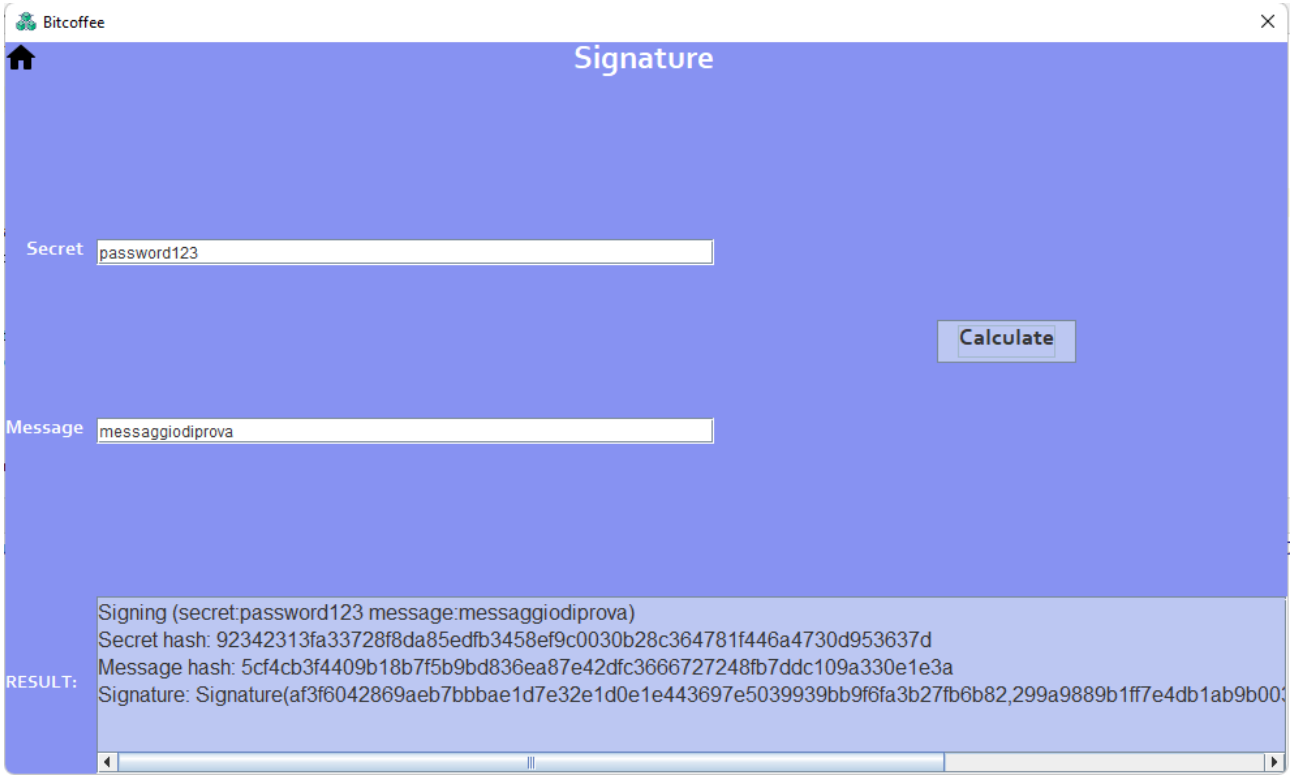
Innanzitutto, la prima finestra sviluppata ed anche la prima con cui l'utente interagirà è la dashboard: nient'altro che una semplice homepage in cui sono presenti diversi pulsanti il cui click su di essi porterà ad aprire una finestra diversa in base alla funzionalità che si intende utilizzare.

Dashboard (homepage) di bitcoffee

Ogni pulsante rappresenta il collegamento a una delle otto funzioni descritte nel capitolo precedente. Le finestre che verranno aperte alla pressione dei bottoni sono molto simili tra loro a livello visivo: tutte hanno a disposizione dei campi su cui scrivere gli input, un pulsante per avviare l'operazione e un bordo su cui verrà stampato l'output dell'operazione invocata.
In alto a sinistra è presente un tasto "home" che consente di tornare al menù principale in qualsiasi momento per poter passare da una funzionalità a un'altra.

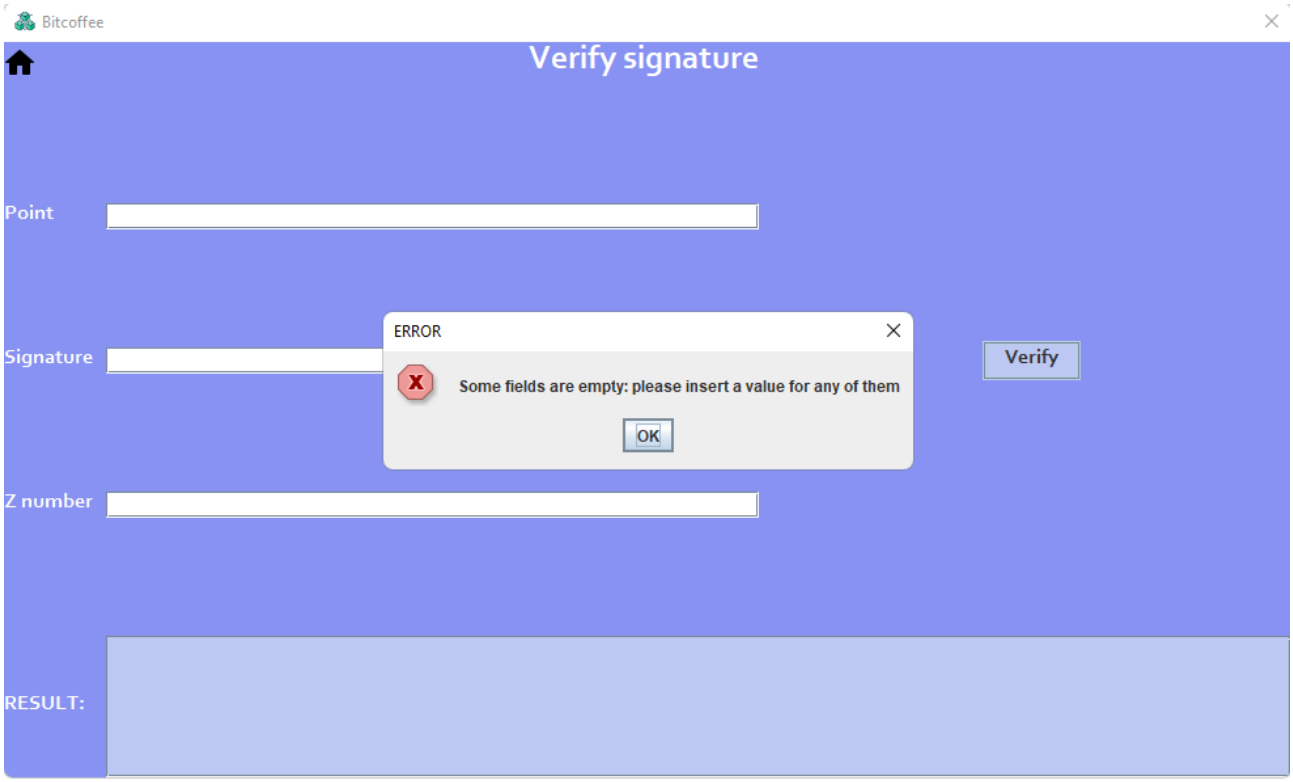


Esempio di finestra interattiva: ecco ciò che l'utente visualizza alla pressione del tasto "Signature"



Dopo aver compilato i due campi e cliccato sul pulsante "calcola", i risultati verranno visualizzati nell'area in basso. Oltre alla firma, vengono anche mostrati gli hash delle due rispettive stringhe utilizzate per la creazione della firma.

Per avere una GUI completa e ottimizzata, sono state gestite le varie eccezioni che possono ricorrere sia in base agli input inseriti dall'utente sia in base ad eventuali errori durante la computazione e lo svolgimento della funzione richiesta.
Tra le più rilevanti è presente quella del controllo dei campi di input per evitare parametri vuoti/nulli.

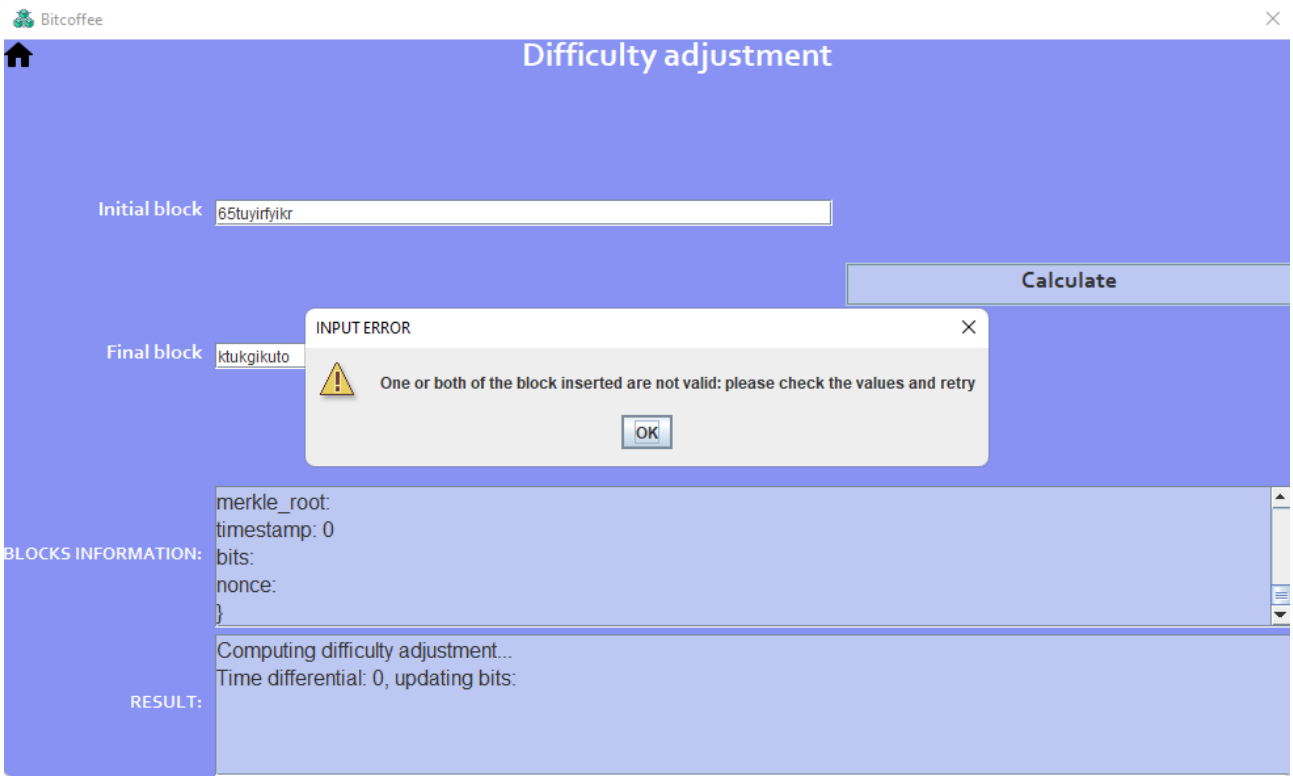


Se uno o più campi non contengono nessun testo, non sarà possibile invocare l'operazione desiderata e verrà generato un pop-up di errore

```
verifyButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String sec= textFieldSec.getText();
        String der= textFieldDer.getText();
        String z= textFieldZ.getText();
        if (sec.isEmpty() || der.isEmpty() || z.isEmpty()) {
            JOptionPane.showMessageDialog(parent,
                    message: "Some fields are empty: please insert a value for any of them",
                    title: "ERROR", JOptionPane.ERROR_MESSAGE);
        } else {
            cmd_verifySign(sec, der, z);
        }
```

Implementazione del popup e arresto dell'avvio del comando nel caso in cui ci siano campi vuoti

Tra le tante funzionalità inserite e utilizzabili, molte hanno bisogno di stringhe di input che siano di una certa tipologia per poter correttamente funzionare; non basta digitare due stringhe randomiche come accade ad esempio per l'operazione Signature. Per questo motivo, è stato anche gestito il controllo del formato degli input inseriti dall'utente.

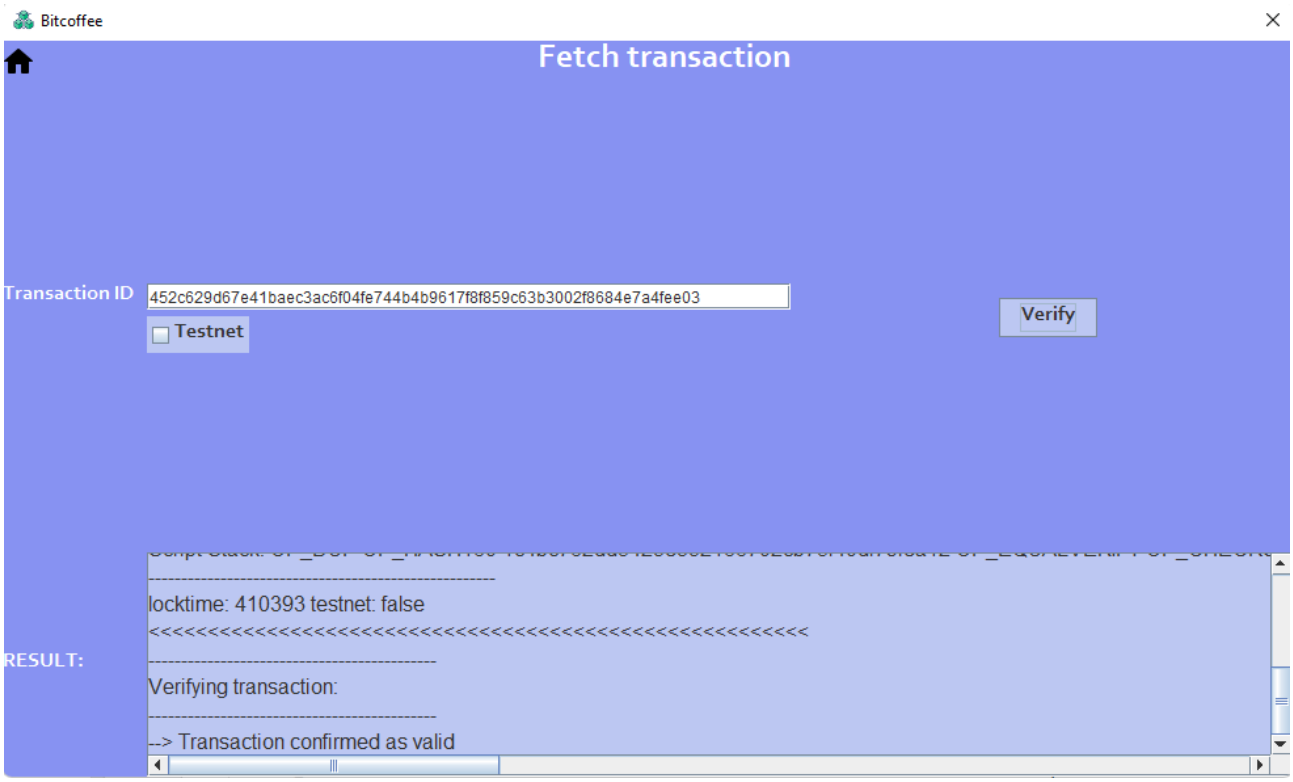


Nonostante entrambi i campi siano compilati, la funzionalità Difficulty Adjustment ha bisogno come parametri due stringhe che rappresentano l'esadecimale seriale del blocco iniziale e del blocco finale

```
    var time_diff = last_block.getTimestamp() - first_block.getTimestamp();
    textAreaResult.append("Time differential: " + time_diff + ", updating bits: " + Kit.bytesToHexString(first_b
    var new_bits : byte[] = Block.computeNewBits(first_block.getBits(), time_diff);
    textAreaResult.append("New bits: " + Kit.bytesToHexString(new_bits));
} catch (Exception e) {
    JOptionPane.showMessageDialog( parentComponent: this,
            message: "One or both of the block inserted are not valid: please check the values and retry",
            title: "INPUT ERROR", JOptionPane.WARNING_MESSAGE);
}
```

Il codice per implementare questo controllo consiste principalmente in un blocco try catch cercando di catturare le varie eccezioni che possono essere generate da input utente non validi

Dove è necessario specificare la rete in cui avviare il tool desiderato, è stato utilizzato un check box: se viene spuntato, allora l'utente decide di utilizzare la rete testnet; altrimenti, l'istruzione verrà avviata facendo riferimento alla rete mainnet.



La transazione inserita esiste solo in mainnet: non spuntando il checkbox, la verifica viene effettuata nella rete mainnet. Di conseguenza, la transazione viene confermata come valida.

4 Risultati e conclusioni

Grazie a questo elaborato, il progetto bitcoffee adesso ha a disposizione un'interfaccia grafica semplice ed intuitiva che permette di testare le funzionalità attualmente sviluppate. Nel caso in cui venissero implementate altre istruzioni nel futuro, si possono facilmente integrare nella GUI semplicemente aggiungendo dei pulsanti nella homepage e personalizzando la finestra dell'istruzione in questione con gli input necessari ed eventuali aree di output per visualizzare il risultato.