Università Parthenope di Napoli

Progetto esame "Programmazione 3 e Lab."

Dumbchain

Alessio Maddaluno (0124/1455)

Docenti:
Prof. Angelo Ciaramella, Prof Raffaele Montella

Indice

1 Descrizione Progetto **3**
1.1 Definizione dei Requisiti 3
1.2 Interfaccia Grafica 4

2 Design Patterns **5**
2.1 Strategy 5
2.2 Builder 6
2.3 Memento 7
2.4 MVC 8

3 Class Diagram **9**

4 Dettagli implementativi **10**
4.1 Mining 10
4.2 Blockchain validation 11
4.3 SHA-256 12
4.4 Merkle Tree Root 12

5 Istruzioni per l'esecuzione **14**

Capitolo 1

Descrizione Progetto

Lo scopo di questo progetto è la realizzazione di un package che emuli il comportamento di una blockchain. Trattandosi di un progetto a scopo puramente didattico, tale implementazione prevederà solo gli aspetti essenziali del funzionamento di tale struttura, come la validazione dell'intera catena, costruzione di blocchi e mining di quest'ultimi. Essendo quindi un'estrema semplificazione, da qui in avanti anziché definirla "blockchain" verrà indicata come "dumbchain".

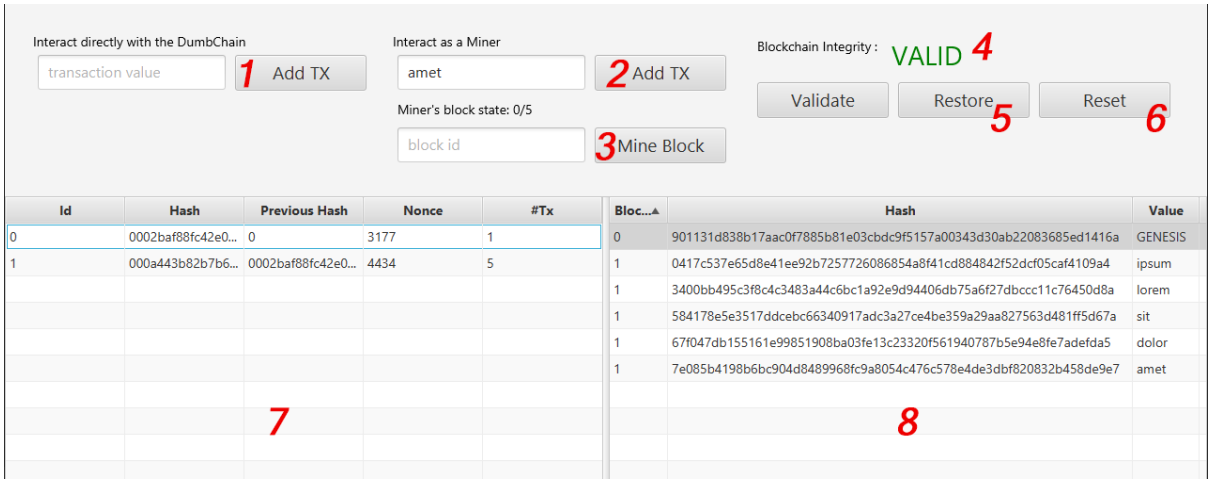
1.1 Definizione dei Requisiti

La dumbchain è una semplificazione di una blockchain. E' dotata di diversi aspetti e caratteristiche:

- Mantiene in memoria una lista di blocchi crittograficamente consecutivi: il blocco nella posizione i contiene l'hash del blocco nella posizione *i-1*;
- Fornisce un sistema di validazione dell'intera catena (*blockchain validation*), stabilendo se quindi il sistema è integro oppure ha subito delle alterazioni;
- Effettua l'hash di blocchi e transazioni utilizzando SHA-256;
- Fornisce un sistema di mining dei blocchi nella catena, utilizzando appositi miner. L'algoritmo predefinito per il mining dei blocchi è il Proof of Work;
- Consente di interagire direttamente con essa o tramite l'ausilio di una pool di transazioni gestita dai miner;
- Consente di ripristinare un suo stato precedente;
- Offre una serializzazione degli elementi che la compongono, facilitando operazioni di diagnostica e approvvigionamento di dati;

1.2 Interfaccia Grafica

Per completezza è fornita un' interfaccia grafica che consente di interagire con la dumbchain. Tale GUI è **completamente indipendente** dal package principale, dando così ampio spazio di estensione a chiunque voglia in futuro partecipare a tale progetto. Tale interfaccia fornisce gli strumenti base per interagire con la dumbchain effettuando varie operazioni, vedendo in tempo reale il suo contenuto e la sua validità. In seguito una breve descrizione delle sue funzionalità:



1. Inserisce una tx direttamente all'intero della blockchain. Non effettua mining e non effettua un controllo d'integrità;
2. Inserisce una tx nella pool del miner. Quando verrà raggiunto il threshold del blocco, il miner provvederà a inserire il nuovo blocco all'interno della dumbchain;
3. Dato l'id di un blocco, il miner provvederà a minarlo. **Non viene fatto nessun controllo sull'integrità della chain**;
4. Mostra lo stato aggiornato all'ultimo controllo della validazione della chain;
5. Ripristina lo stato precedente della dumbchain;
6. Effettua un reset totale della catena;
7. Mostra l'insieme dei blocchi presenti sulla dumbchain;
8. Mostra l'insieme delle transaction presenti all'interno dei blocchi;

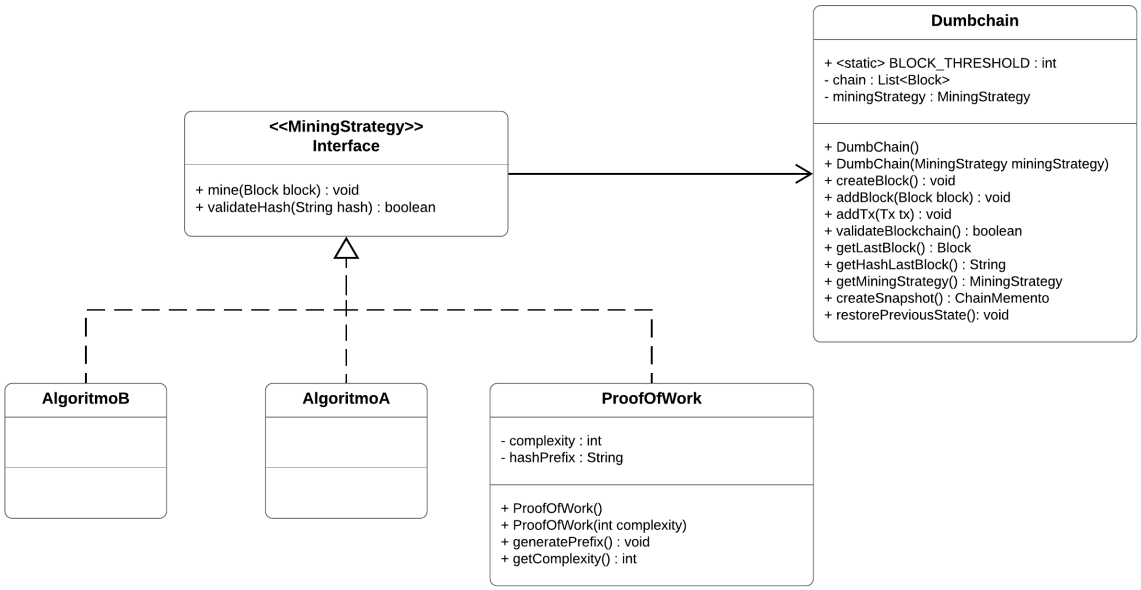
Capitolo 2

Design Patterns

Per la realizzazione di tale progetto sono stati utilizzati quattro pattern: tre per il package principale e uno per la GUI.

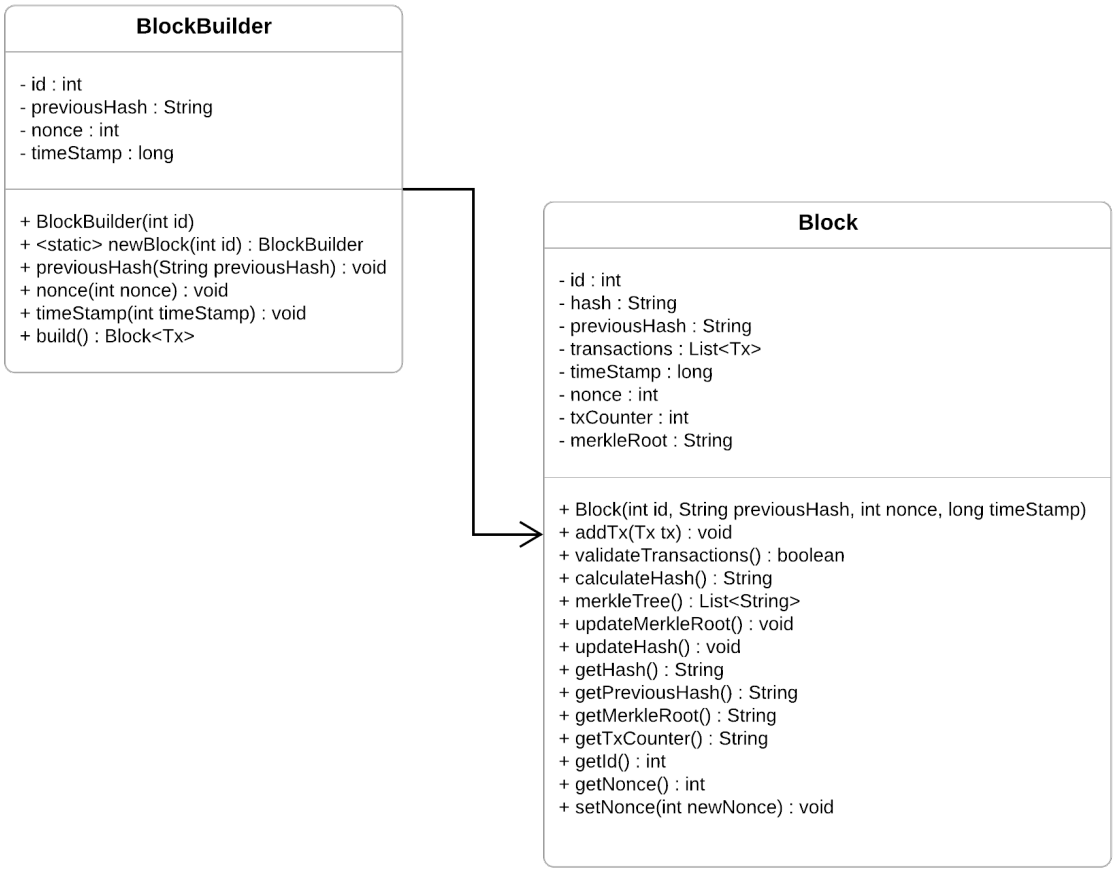
2.1 Strategy

Essendo una semplificazione di una blockchain, è importante per la Dumbchain essere completamente indipendente dal suo algoritmo per il mining. Questo perchè è possibile che si vogliano effettuare considerazioni utilizzando algoritmi di automining o sperimentarne altri che richiedono particolari condizioni da parte dei blocchi. Il miglior pattern per isolare tale comportamento e cambiarlo eventualmente all'occorrenza è il pattern **Strategy**.



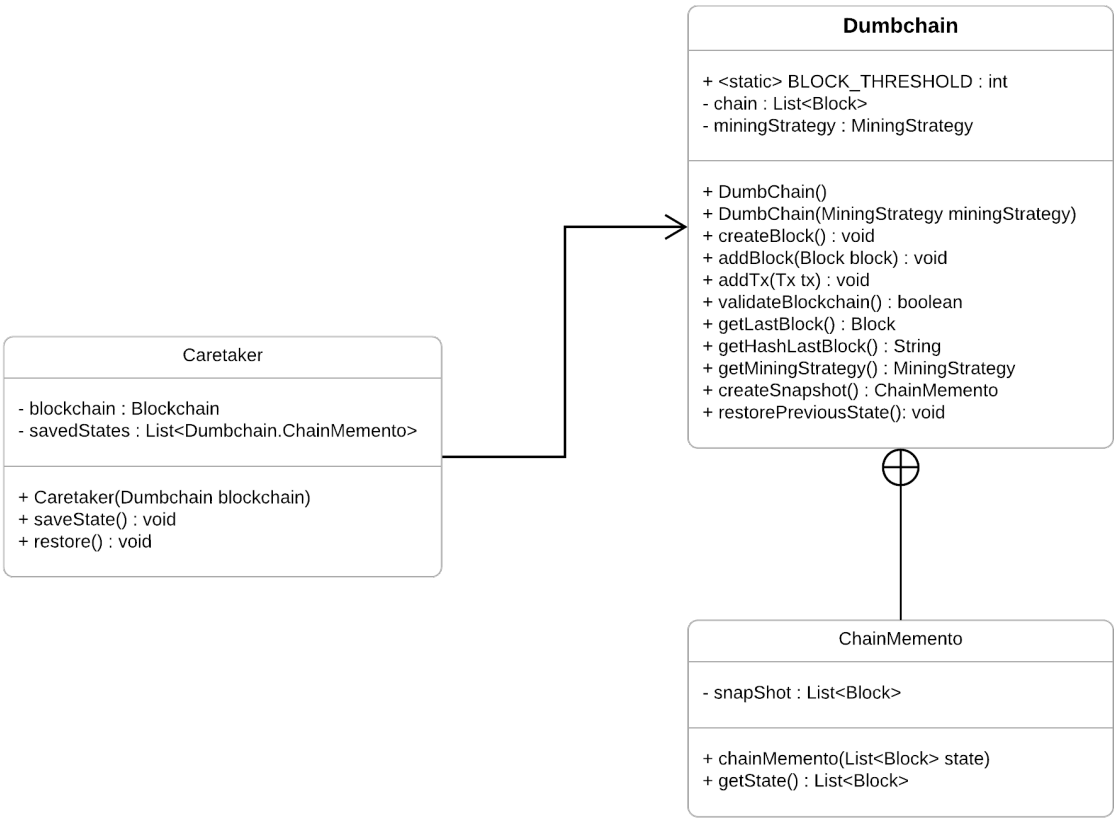
2.2 Builder

Essendo un blocco costituito da molte informazioni e dovendo gestire eventualmente parametri di default o un'inizialiazzazione personalizzata di tali valori, un pattern che aiuta molto e rende semplicissima la creazione - in questo caso di un blocco - è il pattern **Builder**.



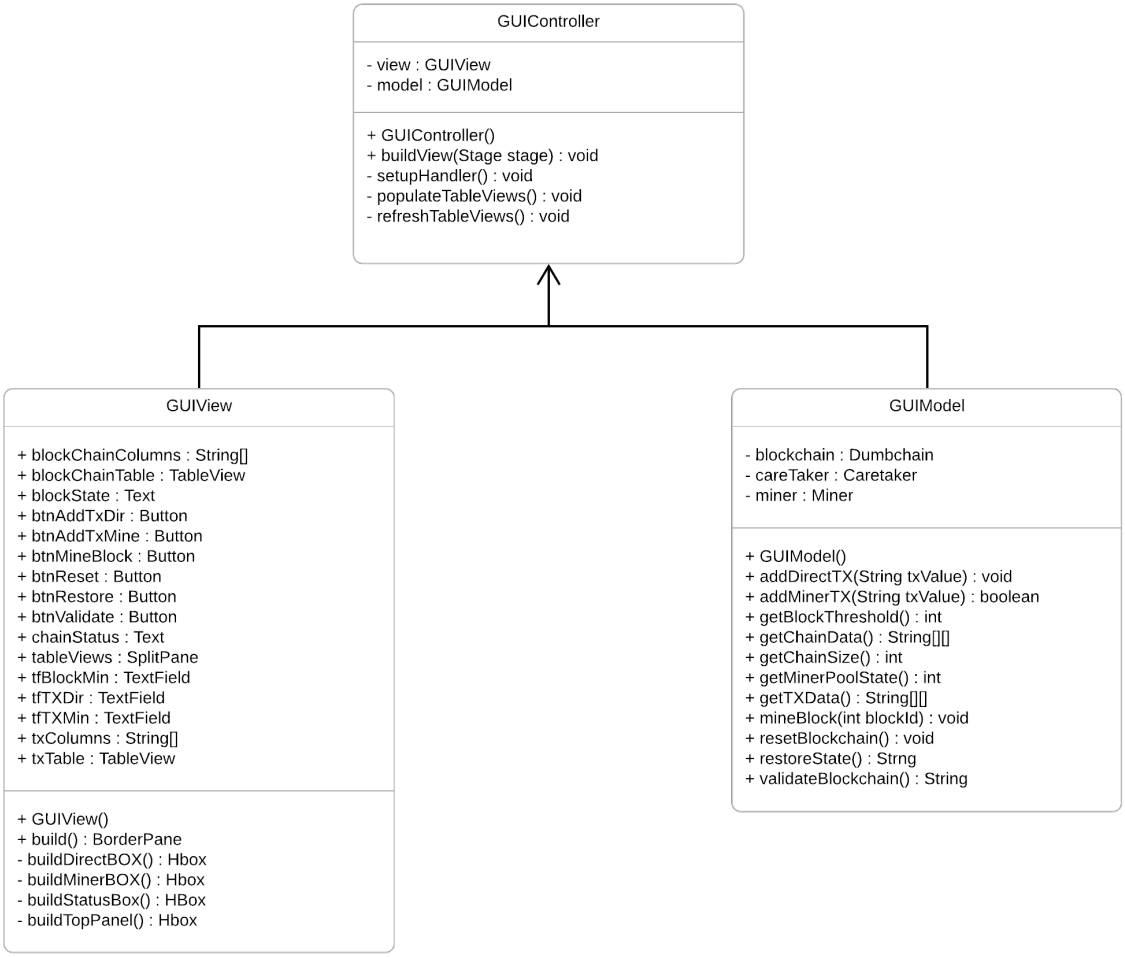
2.3 Memento

La possibilità di ripristinare uno stato precedente della dumbchain è una caratteristica necessaria. Essendo questo un progetto che ha come scopo quello di emularne il comportamento e quindi che consente di inserire blocchi non minati o non integri all'interno della blockchain, il pattern per eccellenza per ripristinare uno stato precedente è il **Mememento**.



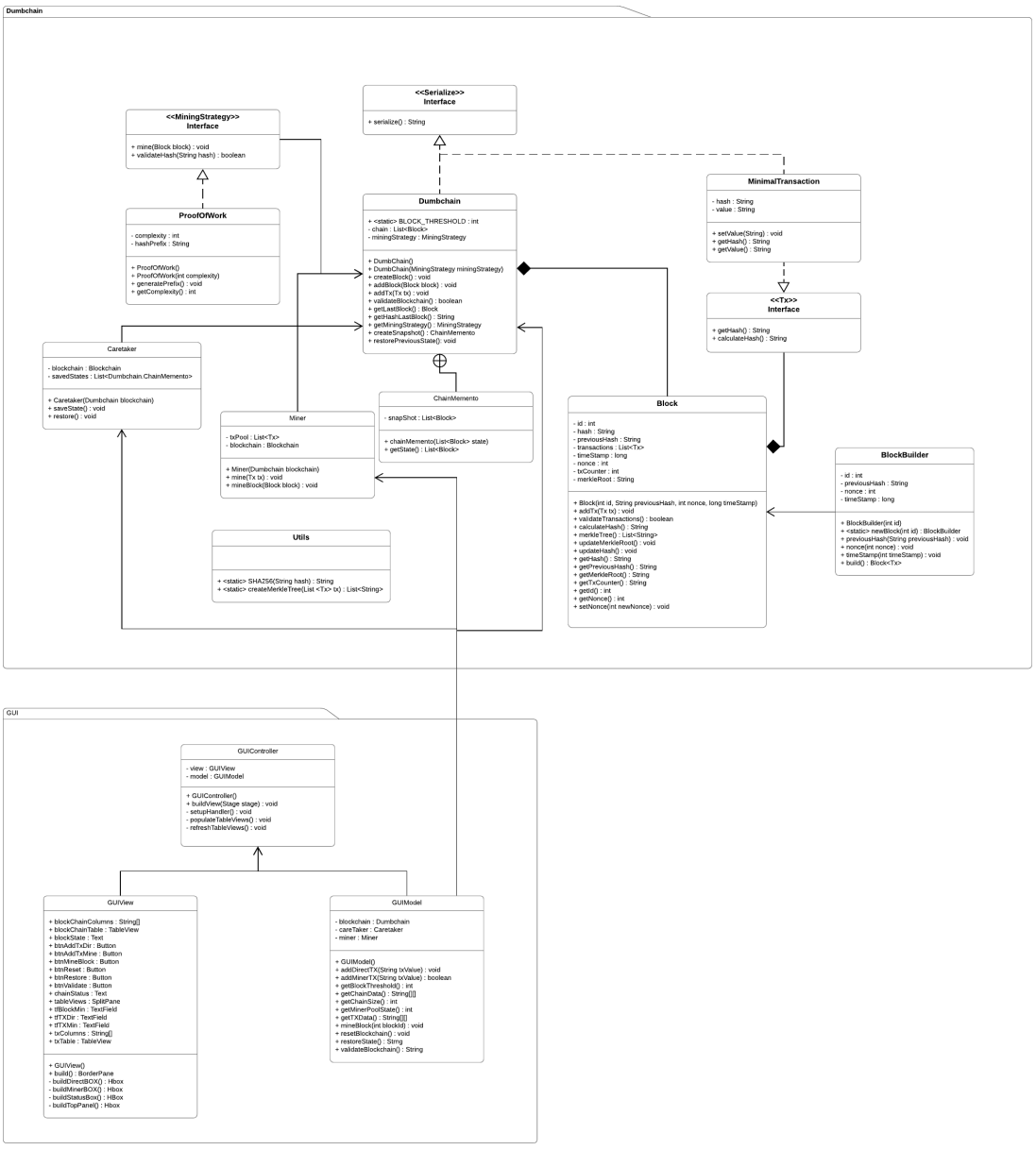
2.4 MVC

Nonostante la GUI non sia una componente necessaria per il funzionamento della dumbchain ma offra solo uno strumento per testarne le funzionalità, dovendo interagire con un package esterno e offrire un interfaccia utente, un buon pattern per suddidere tali compiti è l'**MVC**.



Capitolo 3

Class Diagram



Capitolo 4

Dettagli implementativi

4.1 Mining

Anche se perfettamente estendibile, il mining presente nella Dumbchain ha come algoritmo di default un **Proof Of Work** di *complessità 3*. Proof of work è un algoritmo computazionalmente punitivo che consente ad un' entità di "minare" un blocco i e renderlo quindi permanente all'interno della dumbchain. Il suo funzionamento è molto semplice ed è basato su una costante iterazione di poche operazioni:

1. Calcola l'hash del blocco i
2. Il prefisso dell'hash ottenuto ha n zeri consecutivi?
    - Se si, inserisci il blocco nella dumbchain
    - Se no, incrementa il suo valore di **nonce** e torna al punto 1.

Dove n è la complessità dell'algoritmo, quindi il numero di zeri che l'hash richiede per essere considerato valido. Nel caso della dumbchain tale valore è come precedentemente detto 3, ne consegue che un blocco che non ha 3 zeri consecutivi come prefisso nel suo hash è da considerarsi non valido.

```
public void mine(Block block)
{
 /* The program iterates until doesnt get a correct hash for
    the block. The only difference between the iterations is
    the nonce value */
 while(!block.getHash().substring(0,complexity).equals(this.hashPrefix)
  {
      int nonce = block.getNonce();
      block.setNonce(++nonce);
      block.updateHash();
  }
}
```

4.2 Blockchain validation

Data una blockchain nell'istante temporale t, è necessario stabilire se i blocchi che la costituiscono siano integri e rispettino i requisiti della dumbchain. In particolare per ogni blocco deve essere controllata:

- L'integrità del suo hash;
- L'integrità dell'hash del suo predecessore;
- La condizione di mining imposta dell'algoritmo;

Se anche una di queste condizioni non è verificata, l'**intera** dumbchain è da considerarsi non integra.

```
public boolean validateBlockchain() {

 Block currBlock;
 Block preBlock;

 // Genesis control
 Block genesisBlock = chain.get(0);
 if
    (!genesisBlock.getHash().equals(genesisBlock.calculateHash()))
    // lancia eccezione
 if
    (!this.miningStrategy.validateHash(genesisBlock.getHash()))
     // lancia eccezione

  // Check all the blocks
 for (int i = 1; i < chain.size(); i++)
 {
   currBlock = chain.get(i);
   preBlock = chain.get(i - 1);
   // Check if the block hash is correct
   if(!currBlock.getHash().equals(currBlock.calculateHash()))
        // lancia eccezione
    //Check if the previous block hash is correct
   if(!currBlock.getPreviousHash().equals(preBlock.getHash()))
        // lancia eccezione
    //Check if the block is mined
    if(!miningStrategy.validateHash(currBlock.getHash()))
        // lancia eccezione
 }
 return true;
}
```

4.3 SHA-256

Tutte le operazioni di hashing effettuate sulla dumbchain utilizzano come algoritmo SHA256. Tale algoritmo opera su blocchi di 512 bit (16 parole da 32 bit).
Dopo aver effettuato un opportuno padding se necessario il messaggio è diviso in blocchi e processsati sequenzialmente utilizzando la funzione di compressione dell'algoritmo. Il risultato di tale operazione produce una stringa alfanumerica.

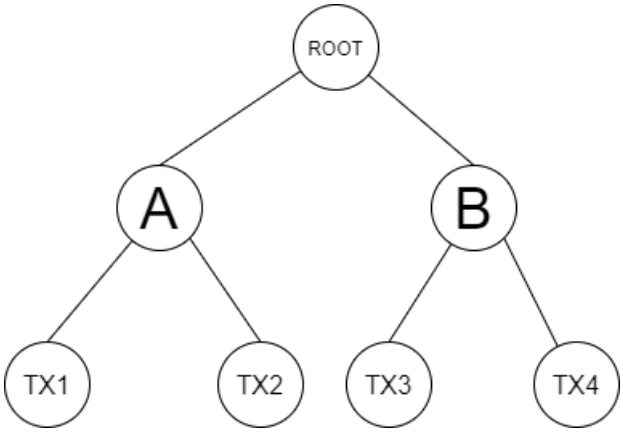
```
public static String SHA256(String data) {
  StringBuffer sb = new StringBuffer();
  try {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(data.getBytes());
    byte[] byteData = md.digest();
    for (int i = 0; i < byteData.length; i++)
      sb.append(Integer.toString((byteData[i] & 0xff) +
         0x100, 16).substring(1));

    }catch (Exception e) {
      e.printStackTrace();
    }
   return sb.toString();
}
```

I crediti di tale implementazione vanno all'utente thatman di StackOverflow:
https://stackoverflow.com/a/44436308

4.4 Merkle Tree Root

La radice di un Merkle Tree è basata su un albero di hash calcolate dalla transazioni memorizzate in un blocco:



In questo esempio l'albero è visto come una lista di nodi: TX1,TX2,TX3,TX4,A,B,ROOT. Le foglie sono gli hash delle transazioni, i nodi interni l'hash della combinzione degli hash dei nodi sottostanti. Tale struttura consente di memorizzare un solo valore di validazione dell'intero blocco.

```
public static List<String> createMerkleTree(List<Tx>
   elements) {
  ArrayList<String> tree = new ArrayList<>();

  for (Tx t : elements) {
    t.getHash();
    tree.add(t.getHash());
  }

  int levelOffset = 0;
  for (int levelSize = elements.size(); levelSize > 1;
     levelSize = (levelSize + 1) / 2) {
    for (int left = 0; left < levelSize; left += 2) {
      int right = Math.min(left + 1, levelSize - 1);
      String tleft = tree.get(levelOffset + left);
      String tright = tree.get(levelOffset + right);
      tree.add(Utils.SHA256(tleft + tright));
    }

   levelOffset += levelSize;
  }
  return tree;
}
```

Un ENORME ringraziamento va al progetto BitcoinJ per l'implementazione di tale metodo. https://github.com/bitcoinj/bitcoinj

Capitolo 5

Istruzioni per l'esecuzione

È possibile liberamente compilare il software da un IDE ed eseguirlo direttamente. Qualora si volesse utilizzare l'eseguibile, senza quindi la necessità di dover utilizzare un IDE è necessario includere nel jar il path delle lib di JavaFX, questo perchè dopo la versione 11, non sono più inserite all'interno dell'SDK.

Per poter quindi eseguire DumbchainGUI è necessario da terminale recarsi nella directory in cui è presente il .jar e lanciarlo con questo comando:

```
java --module-path PATH_LIB_JAVAFX
--add-modules javafx.controls,javafx.fxml,javafx.graphics,javafx.web
-jar DumbchainGUI.jar
```

Dove PATH_LIB_JAVAFX è il path della directory lib di JavaFx.