DMBLOCK zadanie 1

Fakulta informatiky a informačných technológií Slovenskej technickej univerzity

Predmet: Digitálne meny a Blockchain

Zadanie 1 - MyBlockchain

Autor: Roman Bitarovský

Cvičiaci: Viktor Valaštín

21. 03. 2022

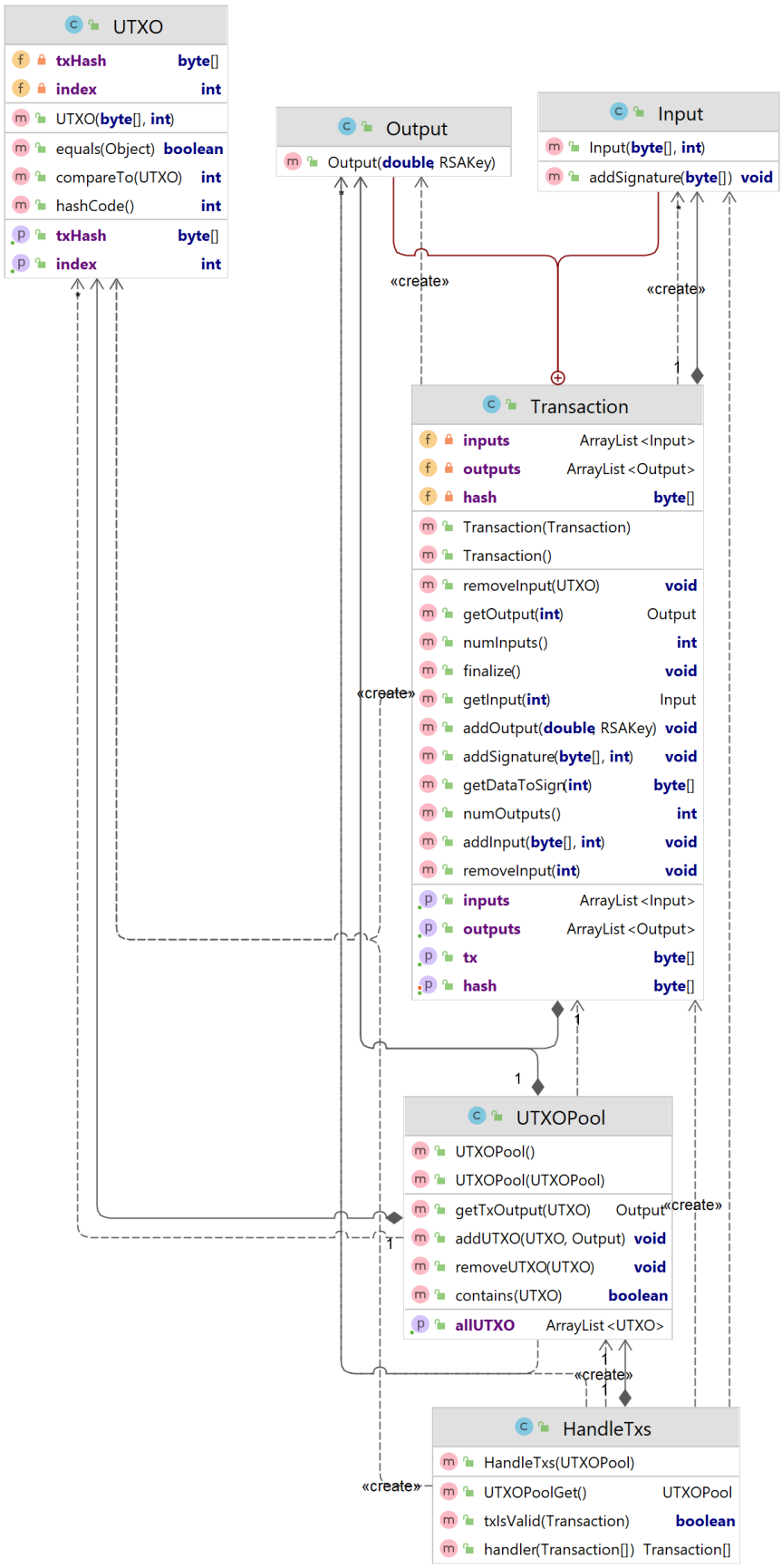
Disclaimer

Uvedené ukážky kódu v tejto dokumentácii nemusia byť 100% zhodné a syntakticky správne s reálnym kódom a boli zjednodušené pre lepšiu čitateľnosť alebo z dôvodu formátovania.

Obsah

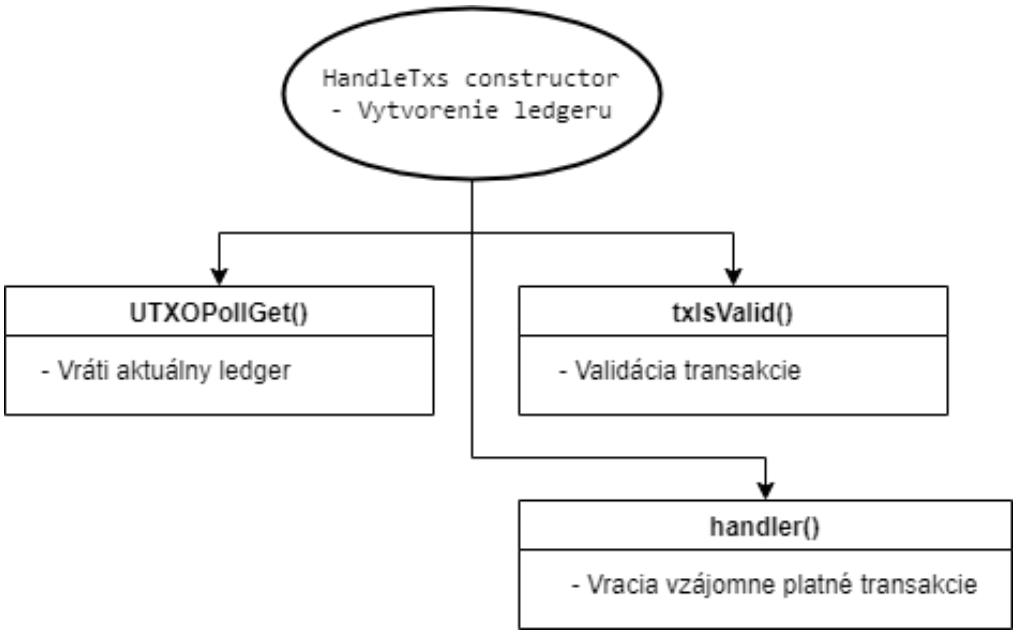
Fáza 1

UML diagram Fáza 1


UTXO
txHash byte[]
index int
UTXO(byte[], int)
equals(Object) boolean
compareTo(UTXO) int
hashCode() int
txHash byte[]
index int
Output
Output(double, RSAKey)
Input
Input(byte[], int)
addSignature(byte[]) void
«create»
«create»
1
Transaction
inputs ArrayList<Input>
outputs ArrayList<Output>
hash byte[]
Transaction(Transaction)
Transaction()
removeInput(UTXO) void
getOutput(int) Output
numInputs() int
finalize() void
getInput(int) Input
addOutput(double, RSAKey) void
addSignature(byte[], int) void
getDataToSign(int) byte[]
numOutputs() int
addInput(byte[], int) void
removeInput(int) void
inputs ArrayList<Input>
outputs ArrayList<Output>
tx byte[]
hash byte[]
«create»
1
1
UTXOPool
UTXOPool()
UTXOPool(UTXOPool)
getTxOutput(UTXO) Output
addUTXO(UTXO, Output) void
removeUTXO(UTXO) void
contains(UTXO) boolean
allUTXO ArrayList<UTXO>
«create»
1
1
«create»
HandleTxs
HandleTxs(UTXOPool)
UTXOPoolGet() UTXOPool
txIsValid(Transaction) boolean
handler(Transaction[]) Transaction[]
«create»

Opis funkcionality `HandleTxs`

Trieda `HandleTxs`

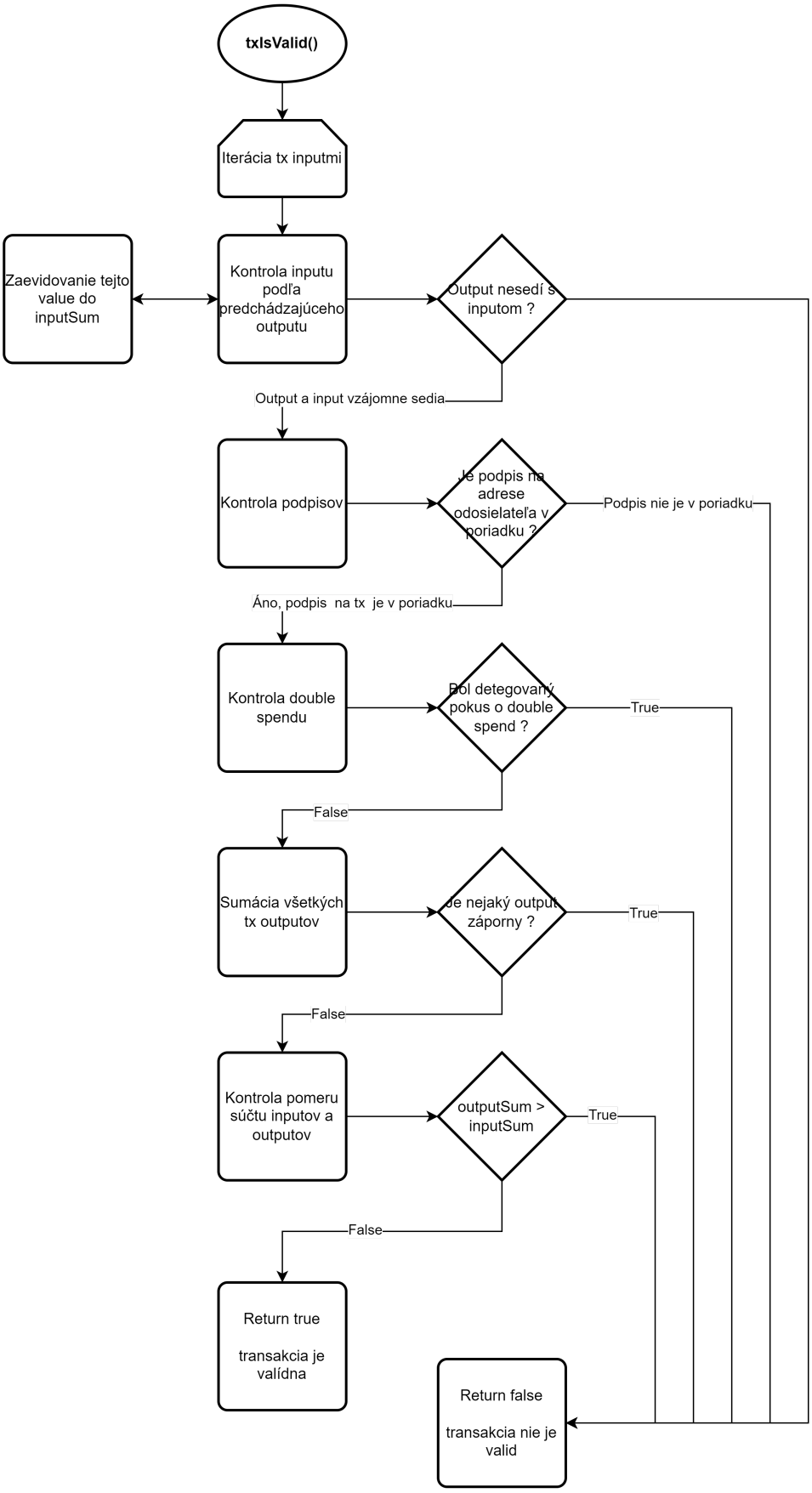


Blokový diagram triedy HandleTxs

Trieda `HandleTxs` okrem konštruktoru obsahuje ďalšie tri metódy.

- `UTXOPoolGet()` → pre získanie aktuálneho `ledger`-u (teda UTXO poolu outputov)
- `txIsValid()` → metóda na validáciu transakcie na základe 5 podmienok
- `handler()` → na vytvorenie množiny vzájomne platných transakcií

Opis funkcionality metódy `txIsValid()`



Blokový diagram metódy txIsValid()

Metóda kontroluje valídnosť transakcie na základe piatich kritérií. Ak je počas kontroly akékoľvek kritérium porušené, metóda vracia false. Ak kontrola kritérií prebehne v poriadku, transakciu považujeme za valídnu a metóda vráti hodnotu true.

Metóda iteruje inputmi danej transakcie a pre každý jednotlivý input skontroluje kritéria.

Kritérium jedna:

Prvé kritérium je aby aktuálny ledger (aktuálny UTXOpool) obsahoval daný input, to sa zisťuje podľa predchádzajúcich outputov.

```
// Pridaj predchadzajucu hodnotu do UTXOpool
UTXO unspentTransaction = new UTXO(txInput.prevTxHash, txInput.outputIndex);

// (1) Aktuálne UTXO musi obsahovat predchadzajuci output
if (this.ledger.contains(unspentTransaction) == true)
  // ak obsahuje je to v poriadku a pokracuje sa dalej
else
  // ak neobsahuje
  return false
```

Kritérium dva:

Druhým kritériom pre správnosť transakcie je správnosť podpisov transakcie. V našom prípade overujeme podpis outputu z ktorého vznikol náš input.

To sa overí nad adresou outputu pomocou metódy `verifySignature()` poskytnutej triedou `RSAKeyPair` z `rsa.jar`

```
ledger.getTxOutput(unspentTx).address.verifySignature(tx.getDataToSign(i), txInput.signature)
```

Ak je podpis v poriadku pokračuje sa ďalej ak nie, metóda končí a vracia hodnotu false nakoľko transakcia nie je valídna.

Kritérium tri:

Tretím kritériom je, že v rámci UTXO nie je pokus o double spend. To overujeme priebežným evidovaním spracovaných outputov predchádzajúcim aktuálnych inputov a kontrolou či už daný input nebol spendnutý. Ak je snaha jeden input v rámci transakcie minúť dvakrát v jeho plnej výške metóda končí vrátením hodnoty false nakoľko by sa jednalo o double spend a teda transakcia nie je valídna.

```
// (3)
if (seenUTXO_set.contains(unspentTransaction) == false) {
  // pridaj transakciu do evidencie akoby pouzitych txs
  seenUTXO_set.add(unspentTransaction);
else
  return false
```

Kritérium štyri:

Medzi ďalšie kritéria patrí kontrola či sa kontrolovaná transakcia nesnaží vytvoriť záporný output. Výstup transakcie so zápornou hodnotou je zakázaný.

To je možné overiť jednoduchou podmienkou:

```
if (output.value < 0)
  return false;
```

Dôležité je takto skontrolovať všetky outputy danej transakcie.

Kritérium päť:

Posledným kritériom je, že súčet inputov danej transakcie musí byť >= ako súčet outputov.

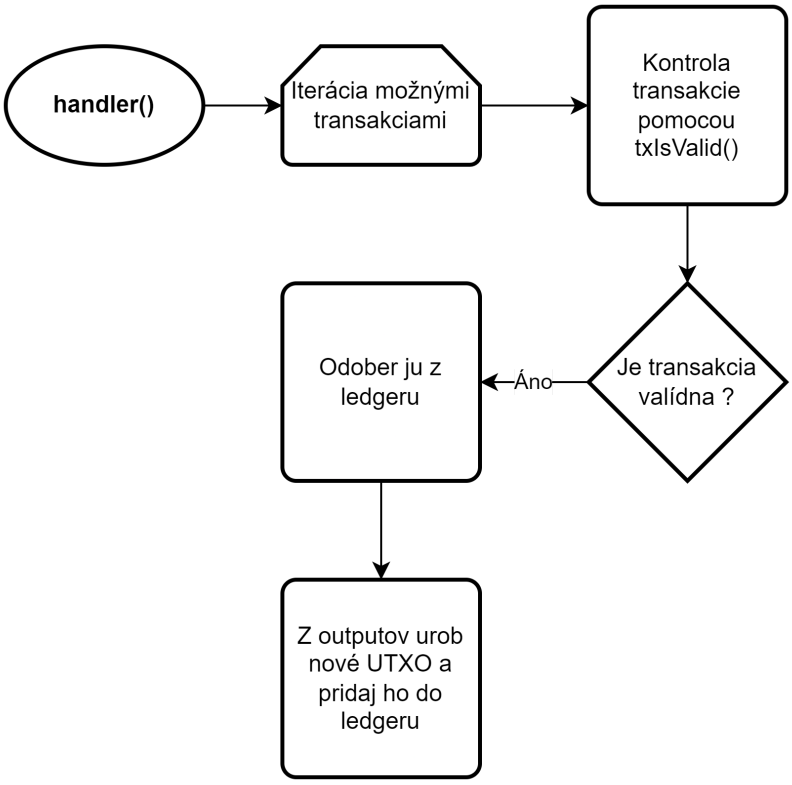
Logicky transakcia nemôže minúť viac ako bolo na jej vstupe.

```
if (outputSum > inputSum)
  return false;
```

Súčet outputov je potrebné spočítať sumarizáciou hodnoty pre všetky outputy.

Opis funkcionality metódy `handler()`

Táto metóda má na vstupe množinu možných transakcií a jej úlohou je vrátiť množinu vzájomne platných a valídnych transakcií.

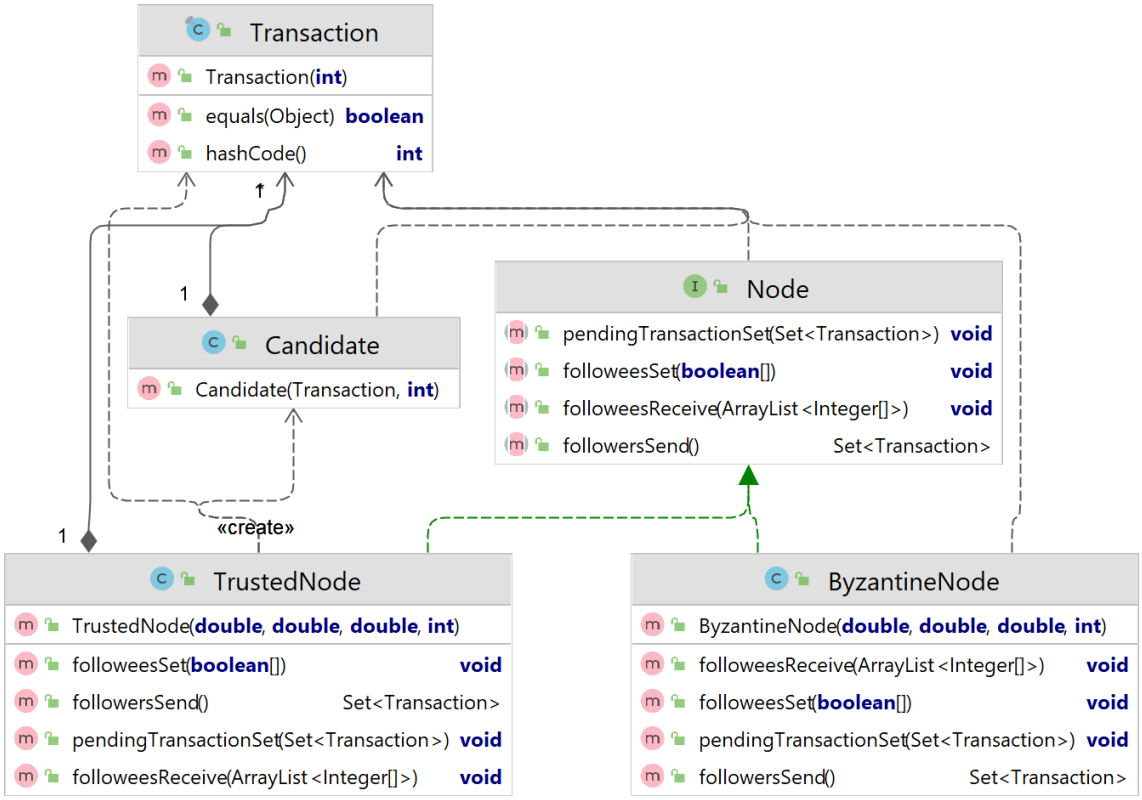


Blokový diagram metódy handler()

Následne po prejdení všetkých možných transakcií máme zoznam valídny a z toho sa na konci metódy vytvorí pole, kt. je následne metódou vrátené.

Fáza 2

UML diagram Fáza 2



Opis funkcionality `TrustedNode`

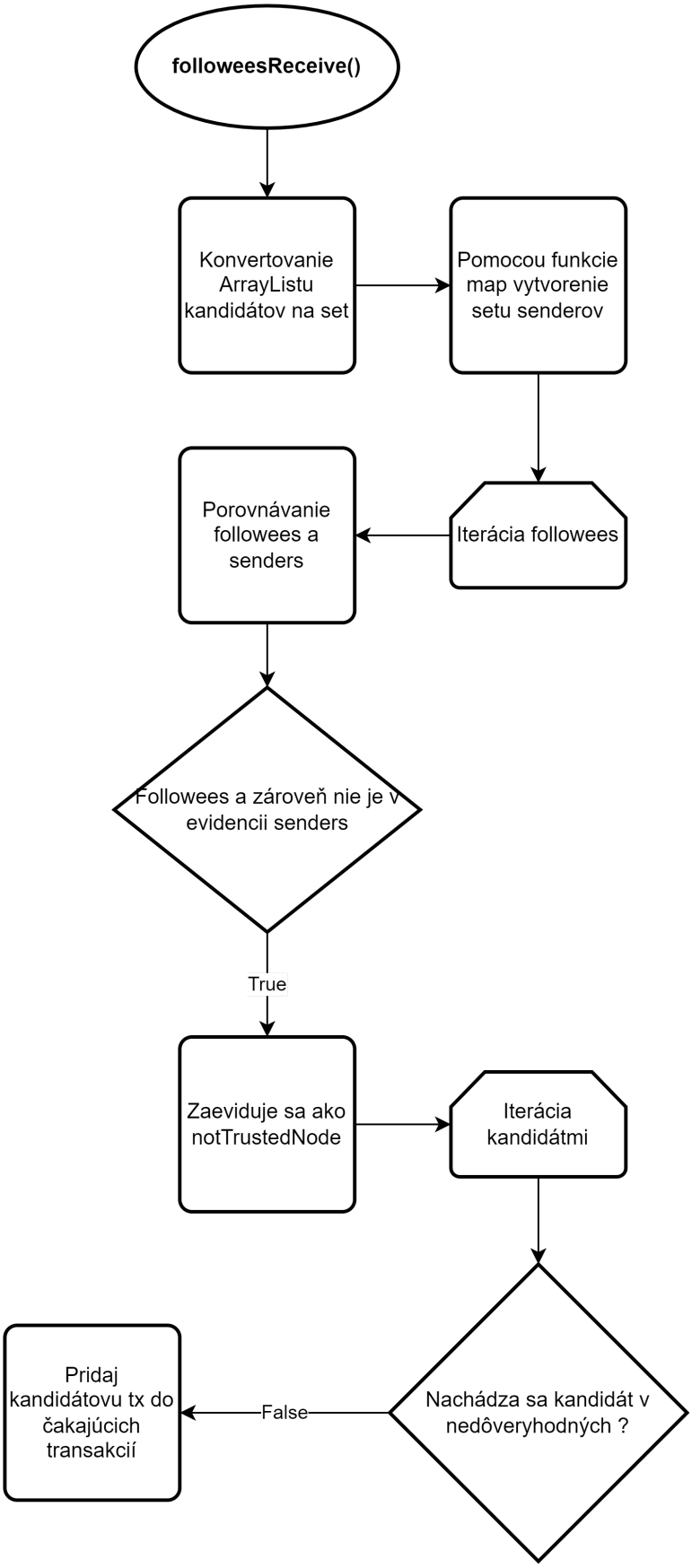
Trieda `TrustedNode`

Okrem konštruktoru obsahuje ďalšie štyri metódy.

- `followeesSet()` → vytvára pole typu boolean pre evidenciu followees
- `pendingTransactionSet()` → spracovanie setu pre čakajúce transakcie
- `followersSend()` → vracia návrhy pre followees a čistí zoznam čakajúcich transakcií
- `followeesReceive()` → spracováva prijatý zoznam kandidátov a snaží sa dosiahnuť konsenzus so sieťou ostatných nodes.

Opis funkcionality metódy `followeesReceive()`

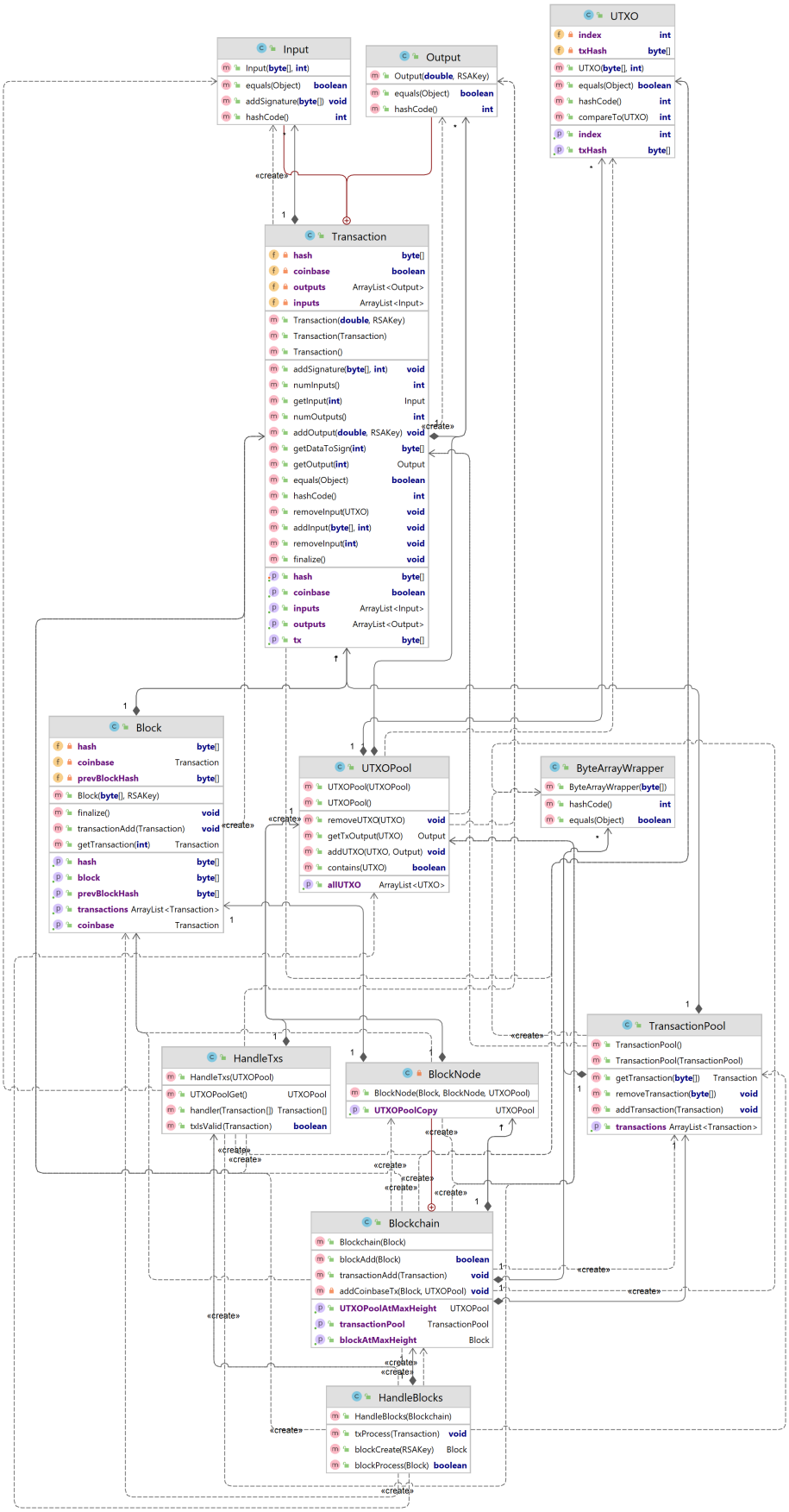
Metóda prijme zoznam kandidátov pričom každý kandidát je reprezentovaný transakciou a id sendera, následne sa vykonáva podľa akcií znázornených diagramom:



Blokový diagram metódy followeesReceive()

Fáza 3

UML diagram Fáza 3


UTXO
index int
txHash byte[]
UTXO(byte[], int)
equals(Object) boolean
hashCode() int
compareTo(UTXO) int
index int
txHash byte[]
Input
Input(byte[], int)
equals(Object) boolean
addSignature(byte[]) void
hashCode() int
Output
Output(double, RSAKey)
equals(Object) boolean
hashCode() int
«create»
Transaction
hash byte[]
coinbase boolean
outputs ArrayList<Output>
inputs ArrayList<Input>
Transaction(double, RSAKey)
Transaction(Transaction)
Transaction()
addSignature(byte[], int) void
numInputs() int
getInput(int) Input
numOutputs() int
addOutput(double, RSAKey) void
getDataToSign(int) byte[]
getOutput(int) Output
equals(Object) boolean
hashCode() int
removeInput(UTXO) void
addInput(byte[], int) void
removeInput(int) void
finalize() void
hash byte[]
coinbase boolean
inputs ArrayList<Input>
outputs ArrayList<Output>
tx byte[]
Block
hash byte[]
coinbase Transaction
prevBlockHash byte[]
Block(byte[], RSAKey)
finalize() void
transactionAdd(Transaction) void
getTransaction(int) Transaction
hash byte[]
block byte[]
prevBlockHash byte[]
transactions ArrayList<Transaction>
coinbase Transaction
UTXOPool
UTXOPool(UTXOPool)
UTXOPool()
removeUTXO(UTXO) void
getTxOutput(UTXO) Output
addUTXO(UTXO, Output) void
contains(UTXO) boolean
allUTXO ArrayList<UTXO>
ByteArrayWrapper
ByteArrayWrapper(byte[])
hashCode() int
equals(Object) boolean
TransactionPool
TransactionPool()
TransactionPool(TransactionPool)
getTransaction(byte[]) Transaction
removeTransaction(byte[]) void
addTransaction(Transaction) void
transactions ArrayList<Transaction>
HandleTxs
HandleTxs(UTXOPool)
UTXOPoolGet() UTXOPool
handler(Transaction[]) Transaction[]
txIsValid(Transaction) boolean
BlockNode
BlockNode(Block, BlockNode, UTXOPool)
UTXOPoolCopy UTXOPool
Blockchain
Blockchain(Block)
blockAdd(Block) boolean
transactionAdd(Transaction) void
addCoinbaseTx(Block, UTXOPool) void
UTXOPoolAtMaxHeight UTXOPool
transactionPool TransactionPool
blockAtMaxHeight Block
HandleBlocks
HandleBlocks(Blockchain)
txProcess(Transaction) void
blockCreate(RSAKey) Block
blockProcess(Block) boolean

Opis funkcionality `Blockchain`

Trieda `Blockchain`

Obsahuje:

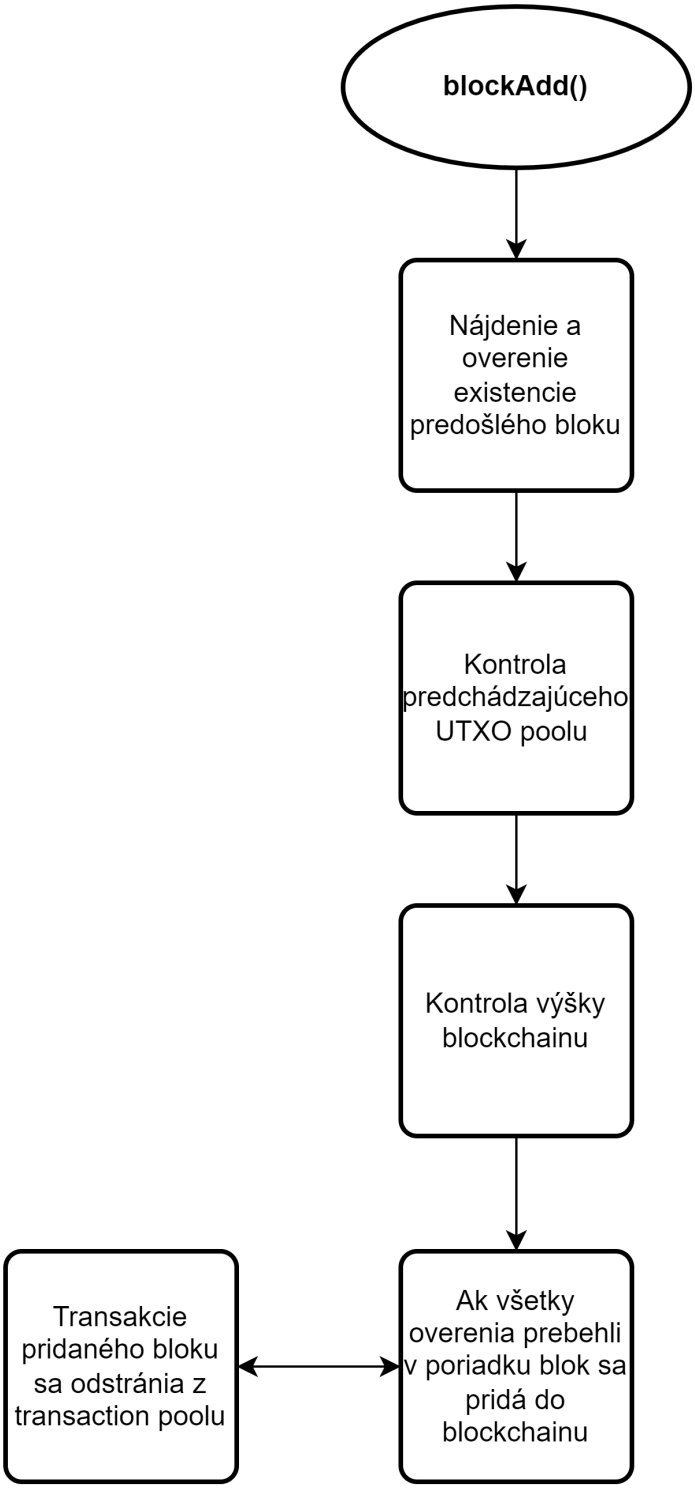
- Privátnu podtriedu `BlockNode()` → Obsahuje dodatočné informácie k bloku
- Konštruktor `Blockchain(Block genesisBlock)` → tento krát zámerne uvádzame konštruktor aj s jeho parametrom nakoľko je to mimoriadne dôležité.
- `getBlockAtMaxHeight()` → vráti najvyšší blok pre blockchain
- `getUTXOPoolAtMaxHeight()` → vráti UTXO pool najvyššieho bloku
- `getTransactionPool()` → vracia transaction pool pre celý blockchain, teda štruktúru kde sú uložené všetky doposiaľ nespracovaný transakcie s potenciálom pre uloženie a uzamknutie do blockchainu.
- `blockAdd()` → pridanie nového bloku do blockchainu
- `transactionAdd()` → pridanie transakcie to blockchainového transaction poolu spomínaného v metóde `getTransactionPool()`
- `addCoinbaseTx()` → moja vlastná metóda pre vytvorenie coinbase transakcie pre blok, coinbase transakcia je transakcia pridávajúca do bloku odmenu za vyťaženie daného bloku.

Opis funkcionality metódy `Blockchain()`

Táto metóda slúži ako konštruktor pre blockchain teda vytvára náš blockchain s potrebnými náležitosťami.

1. Vytvorenie premennej pre blockchain
2. Vytvorenie transaction poolu pre daný blockchain
3. Vytvorenie genesis bloku
4. Pridanie genesis bloku do blockchainu
5. Zaevidovanie genesis bloku ako posledného bloku v blockchaine

Opis funkcionality metódy `blockAdd()`



Blokový diagram metódy blockAdd()

Nájdenie a overenie predchádzajúceho bloku:

```
// zober predosli blok ak nieje null
if (block.getPrevBlockHash() == null){
    return false;
}
```

```
// zober node tohto predosleho bloku ak nieje null
BlockNode parentBlockNode = this.blockChain.get(block.getPrevBlockHash());
if (parentBlockNode == null) {
    return false;
}
```

Kontrola predchádzajúceho UTXO poolu:

```
// skontroluj txs predchadzajuceho bloku
if (handlerTxs.handler(txs).length != txs.length){
    return false;
}
```

Pridanie coinbase transakcie aby bolo možné sprostredkovať odmenu za vyťaženie bloku:

```
UTXOPool utxoPool = handlerTxs.UTXOPoolGet();
// coinbase tx teda vymajnovany BTC treba do bloku pridat ako prvu transakciu
addCoinbaseTx(block, utxoPool);
```

Kontrolu výšky tu uvádzať nebudeme nakoľko sa jedná iba o porovnanie premenných.

Pridanie bloku do blockchainu:

```
BlockNode node = new BlockNode(block, parentBlockNode, utxoPool);
this.blockChain.put(new ByteArrayWrapper(block.getHash()), node)
```

A nakoniec je ešte potrebné transakcie kt. obsahujú pridaný blok odobrať z transaction poolu:

```
// tx kt. su v bloku locknute odober z txPoolu lebo uz su vykonane
block.getTransactions().forEach( (tx) -> {
    this.txPool.removeTransaction(tx.getHash());
});
```

Používateľská príručka

Implementačné prostredie

Projekt je naprogramovaný a spustiteľný v jazyku java za prítomnosti všetkých potrebných knižníc uvedených v importoch jednotlivých tried.

Pre spustenie projektu je potrebné mať nainštalovanú javu.

```
java 17.0.2 2022-01-18 LTS
Java(TM) SE Runtime Environment (build 17.0.2+8-LTS-86)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.2+8-LTS-86, mixed mode, sharing)
```

Ako editor sme používali InteliJ idea od JetBrains.

Spustenie kódu

Pre spuste kódu je potrebné mať vytvorený java projekt (v prípade ak používate InteliJ) a do zložky src nahrať všetky potrebné zdrojové súbory vrátane `rsa.jar` .

Zložku `rsa.jar` je potrebné pridať pomocou pravým klikom na zložku v projekte a vybrať možnosť “Add as library”. Následne by sme mali vedieť kód skompilovať a spustiť.

Záver

Všeobecný komentár

Ako prvé by som chcel povedať, že zadanie dalo celkom zabrať (nečakane). Človek by čakal, že na základe videa k zadaniu a znalostí blockchainu z pohľadu používateľa nebude problém zostrojiť simuláciu blockchainu. Opak bol však pravdou ale na druhú stranu musím povedať, že ma to bavilo.

Čo sa týka technických záležitostí teda Java, to tiež veľmi nepotešilo nakoľko s Javou už dlhšie nerobím ale stále by to mohlo byť aj horšie a mohli by sme to robiť v C/C++ tak ako je je urobený aj BTC. Takže konštatujem, že to bolo v závere stále v pohode a oprášiť Javu tiež nebolo až také zlé.

Teraz k samotnej logike a podstate zadania. Komentára podľa kt. sa máme riadiť pri realizácií zadania by mohli byť aj lepšie a presnejšie. Oni sú síce dosť presné ale kým reálne ten kód neexistuje je ťažké si predstaviť čo reálne máme nakódiť.

Čo som sa naučil

V tomto smere musím zadanie pochváliť, pomohlo mi ešte lepšie pochopiť čo sa v blockchaine deje a aké náležitosti je potrebné riešiť na úrovní kódu.

Zadanie veľmi názorne ukázalo ako sú jednotlivé funkcionality na seba nadviazané a ako jednotlivé funkcionality na sebe stavajú.

Mňa osobne najviac bavila a najviac mi dala fáza tri kde sme si vyskúšali predovšetkým pridávanie nového bloku do reťaze a celkovo skonštruovanie blockchainu od genesis bloku.