Blockchain Test Task

Overview

The task is to implement a very primitive model of blockchain currency, just like Bitcoin, but with many simplifications.

Some useful terms:

- Transaction – a single transaction, that creates ("emits") some coins or moves them between accounts
- Block – a list of transactions
- Block Tree – a tree of Block objects with a single root. Every Block has a single ancestor and unlimited number of descendants.
- Block Chain – the longest chain of Block objects within a Block Tree

Classes

Create 3 classes in your favorite programming language:

- Transaction – represents a single transaction (money emission or transfer) within the system
- Block – holds 1 or more valid transactions
- BlockChain – contains the whole block tree

Transaction Class

Properties:

- Id (integer) – unique transaction id
- Type (integer):
  - 0 – money emission. It means the system creates new "coins" and puts them to the destination account of the transaction ("to" property)
  - 1 – money transfer. It means that within this transaction one account transfers money to some other account.

- From (string) – source account name
- To (string) – destination account name
- Amount (integer) – transaction amount
- Signature (string) – MD5 hex digest of the transaction fields. In order to calculate the signature, you should convert all properties (id, type, from, to, amount) to strings, and concatenate them using a semicolon (":") character. In other words, the signature will be equal to MD5("id:type:from:to:amount"), for example: MD5("1:1:bob:alice:1000") = "2a172cf33d3444b5df7615378e6640e0" – will be a valid signature of "money transfer" transaction from "bob" to "alice" with id 1 and amount of 1000.

Methods:

- Constructor, getId, getType, getFrom, getTo, getAmount, getSignature
- setId(type: integer) – set "id" property.
- setType(type: integer) – set "type" property. Should check if given type is within the allowed range or throw an exception otherwise. If type is "emission", then "from" property should be set to null.
- setFrom(from: string) – set "from" property. The method should perform the following checks:
    - if transaction "type" is "emission" – ignore the passed "from" value and set "from" property to null
    - if passed "from" account is null, is shorter than 2 characters or longer than 10 characters – throw an exception
- setTo(to: string) – set "to" property. The method should perform the following checks:
    - if passed "to" account is null, is shorter than 2 characters or longer than 10 characters – throw an exception
    - if "to" account is the same as the "from" account – throw an exception
- setAmount(amount: integer) – set "amount" property. The method should perform the following checks:
    - if amount is less than zero – throw an exception
- setSignature(signature: string) – set "signature" property. The method should perform the following checks:
    - if passed signature's length is not equal to 32 characters – throw an exception

Block Class

Properties:

- Id (integer) – unique block id
- Transactions (list of Transaction) – a list of transactions within this block

Methods:

- Constructor, getId, getTransactions

- setId(id: integer) – set block id
- validateTransaction(transaction: Transaction): boolean – validate passed transaction. The method should perform the following checks:
    - check if transaction signature is valid - see the signature algorithm description in the Transaction class overview above

  The method should return:
    - true – if the passed transaction is valid
    - false – otherwise
- addTransaction(transaction: Transaction) – add transaction to a list of transactions. The method should perform the following checks before adding a transaction and if the transaction doesn't pass at least one check, it should be simply ignored without throwing any exceptions:
    - validate the transaction using a validateTransaction method
    - check if the number of existing transactions in block is less than 10
    - check if transaction with transaction.id doesn't already exist in the list of transactions in this block

BlockChain Class

Properties:

- Block Tree (create some custom type to store the tree) – a tree of Block objects. Every Block will have only 1 ancestor and the unlimited number of descendants. There will be only 1 tree root – the Block without any parents.

Methods:

- Constructor
- getBlockChain() – the method should return a list of Blocks within the longest chain of Blocks in the Block Tree
- validateBlock(block: Block): boolean – validate passed block. The method should perform the following checks:
    - the block has at least 1 transaction
    - the block with the same id doesn't exist in the "Block Tree" yet

  The method should return:
    - true – if the passed block is valid
    - false – otherwise
- addBlock(parentBlockId: integer, block: Block) – add block to the "Block Tree". The method should perform the following checks before adding a block to the tree, and if at least one of checks fails, the method should simply ignore the block without throwing any exceptions:
    - parentBlockId is null and the root block already exists. As there can be only one root block, we cannot add more blocks like that.
    - parentBlockId refers to a block that doesn't exist in the "Block Tree"
    - adding "block" to the existing "parentBlockId" block will lead to negative balance on some accounts

- getBalance(account: string): integer – get balance of the “account”. The method should perform the following checks:
    - if the “account” is null, is shorter than 2 characters or longer than 100 characters – throw an exception

  The method should return a calculated account balance, using the longest existing chain of blocks in the tree. All transfers to the account or emissions to the account should increase the balance, and all transfers “from” the account should decrease it. If the account didn’t receive any funds, then return zero.

Typical Workflow

The typical workflow for these classes may look as follows:

```
// create 100 coins and transfer them to Bob
Transaction trx = new Transaction();
trx.setId(1);
trx.setType(Transaction.EMISSION);
trx.setTo(“bob”);
trx.setAmount(100);
trx.setSignature(“valid signature goes here”);

Block block = new Block();
block.setId(1);
block.addTransaction(trx);

BlockChain blockChain = new BlockChain();
blockChain.addBlock(null, block);

// bob transfers 50 coins to alice
trx = new Transaction();
trx.setId(2);
trx.setType(Transaction.TRANSFER);
trx.setFrom(“bob”);
trx.setTo(“alice”);
trx.setAmount(50);
trx.setSignature(“valid signature goes here”);

block = new Block();
block.setId(2);
block.addTransaction(trx);

blockChain.addBlock(1, block);
blockChain.getBalance(“alice”); // should return 50
blockChain.getBalance(“bob”); // should return 50
```

Additional Notes

- There is no need to create different exception classes. Using a built-in generic exception class with a custom text message would be enough.
- All methods should be well-documented (method purpose, parameters description)
- There are no additional data validation requirements except mentioned in this document.