Assignment 1 Ledger System

Your source code ready to build (no .class files)

1) The source code can be found in the ZIP file:
    - christopher_jones_assignment1.zip
2) Unzip the ZIP file in a location of your choice
3) Run the following command to build the code:
    - javac com/cscie97/ledger/*.java com/cscie97/ledger/test/*.java
4) Run the following command to test:
    - java -cp . com.cscie97.ledger.test.TestDriver ledger.script

Your data files (including the provided sample)

1) The provided test script is located in the src directory titled ledger.script
2) The testing output files are located in the src directory titled ledger_test.txt

Results of processing your test files

ledger_test.txt

```
>>> Creating a Ledger w/ parameters (test, “test ledger 2020”, “harvard”)

>>> Creating an Account w/ parameters (mary)

>>> Creating an Account w/ parameters (bob)

>>> Creating an Account w/ parameters (bill)

>>> Creating an Account w/ parameters (frank)

>>> Creating an Account w/ parameters (jane)

>>> Get Account Balances w/ parameters (mary)
!ACCOUNT NOT COMMITTED: [The account for "mary" has not been commited yet]

>>> Processing Transaction w/ parameters (1, 1000, 10, “fund account”, master, mary)

>>> Processing Transaction w/ parameters (2, 1000, 10, “fund account”, master, bob)

>>> Processing Transaction w/ parameters (3, 1000, 10, “fund account”, master, bill)

>>> Processing Transaction w/ parameters (4, 1000, 10, “fund account”, master, frank)

>>> Processing Transaction w/ parameters (5, 1000, 10, “fund account”, master, mary)

>>> Get Account Balances w/ parameters (mary)
!ACCOUNT NOT COMMITTED: [The account for "mary" has not been commited yet]

>>> Get all Account Balances w/ parameters ()
!NO BLOCK COMMITTED: [There are no existing committed blocks]
```

ledger_test.txt

```
>>> Processing Transaction w/ parameters (6, 1000, 10, “fund account”, master, bob)

>>> Processing Transaction w/ parameters (7, 1000, 10, “fund account”, master, bill)

>>> Processing Transaction w/ parameters (8, 1000, 10, “fund account”, master, frank)

>>> Processing Transaction w/ parameters (9, 1000, 10, “fund account”, master, mary)

>>> Processing Transaction w/ parameters (10, 1000, 10, “fund account”, master, bob)

>>> Get Account Balances w/ parameters (mary)
3000

>>> Get Account Balances w/ parameters (bob)
3000

>>> Get Account Balances w/ parameters (bill)
2000

>>> Get Account Balances w/ parameters (frank)
2000

>>> Get Account Balances w/ parameters (jane)
0
```

ledger_test.txt UNREGISTERED

```
>>> Get Block details w/ parameters (1)
Hash:09
Transaction:1 > master:mary ($1000)
Transaction:2 > master:bob ($1000)
Transaction:3 > master:bill ($1000)
Transaction:4 > master:frank ($1000)
Transaction:5 > master:mary ($1000)
Transaction:6 > master:bob ($1000)
Transaction:7 > master:bill ($1000)
Transaction:8 > master:frank ($1000)
Transaction:9 > master:mary ($1000)
Transaction:10 > master:bob ($1000)

>>> Get Transaction details w/ parameters (10)
ID:         10
Payer:      master
Receiver:   bob
Amount:     1000
Fee:        10
Note:       “fund account”

>>> Processing Transaction w/ parameters (11, 200, 10, “september rent”, frank, jane)

>>> Get Account Balances w/ parameters (frank)
2000

>>> Get Account Balances w/ parameters (jane)
0
```

ledger_test.txt UNREGISTERED

```
>>> Processing Transaction w/ parameters (12, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (13, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (14, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (15, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (16, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (17, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (18, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (19, 20, 10, “uber”, bob, mary)

>>> Processing Transaction w/ parameters (20, 20, 10, “uber”, bob, mary)

>>> Get Account Balances w/ parameters (jane)
200

>>> Get Account Balances w/ parameters (frank)
1790

>>> Get Account Balances w/ parameters (mary)
3180

>>> Get Account Balances w/ parameters (bob)
2730

>>> Get all Account Balances w/ parameters ()
{frank=1790, bob=2730, mary=3180, bill=2000, master=2147473747, jane=200}

>>> Processing Transaction w/ parameters (21, 5000, 10, “food”, bob, mary)
!INVALID TRANSACTION: [The payer bob has insufficient funds]

>>> Processing Transaction w/ parameters (22, 20, 5, “food”, bob, mary)
!INVALID TRANSACTION: [The payer bob has insufficient funds]
```

```
>>> Validating details w/ parameters ()
Block #1: Valid
Block #2: Valid
```

Any changes that you made to the proposed design and how they continue to support the requirements

1) Account Class

    a. Included the uint function since Java does not support unsigned integers. This function is solely to ensure that no value falls out of the specified range.

2) CommandProcessor Class

    a. While both options are available (one is just commented out), the processCommand logs an error message to the console instead of throwing an exception. The main purpose for this was because throwing the Exception halted the program, and I wanted to capture the error, log it, and then continue running.

3) MerkleTree Class

    a. I included this class to utilize in my Block class to help generate SHA256 hashes each of the individual Blocks.

4) TestDriver Class

    a. I made this class such that it can run either with a script, or directly in the terminal with user-input. This wasn't a specific ask, but it came up during office hours and seemed necessary.

Did the design document help with the implementation?

1) The design document helped considerably, especially when it came to laying out the initial groundwork. There were a few questions that came up along the way brought on a few extra requirements, but other than that, it was very helpful.

How could the design have been better, more straightforward, or made the implementation easier?

1) There were a few requirements asking us to show details stored within a particular object, but not necessarily what was needed to be shown or in what format. It would be great to have some kind of guidance in terms of what the most/least important information is and also what is considered private/public information.