CS 2334: Programming Structures and Abstraction

Project2

Data Manipulation with Ethereum

1. Objective:

The objective of this programming project is to examine reading a file, parsing data, performing calculation on data. After completing the project, students will have an intermediate understanding of reading file and performing calculation on data in Java. Students will also have an intermediate understanding of date and time operations in Java.

2. Project Specification:

2.1. Overall Program Behavior:

A blockchain is a database of transactions that is updated and shared across many computers in a network. Every time a new set of transactions is added, it's called a "block" - hence the name blockchain. Most blockchains are public, and you can only add data, not remove. If someone wanted to alter any of the information or cheat the system, they'd need to do so on the majority of computers on the network. For this project, we will be using a dataset of 100 blocks in the Ethereum blockchain.

We provided Driver.java and ethereumP1data.txt [do not edit these two files]. You are required to update the Blocks class you created for project 1 to create the new methods needed to complete the project based on this project description and Driver.java we provided.

2.2. Read Input File:

Your program is required to read the content of the file provided, ethereumP1data.txt. For this project, we are interested in the timestamp and transaction_count, columns 17 and 18 respectively in the file. You can see more information on the data here: https://ethereum-etl.readthedocs.io/en/latest/schema/ under the section blocks.csv. Your goal is to use this information to fill an ArrayList of Blocks objects stored in your Blocks class. Each Blocks object should at least hold the number and miner from project 1 and the timestamp and transaction_count now as each of these fields will be used later in the project.

2.3. Time Calculations:

The timestamp you read in from the data file is a Unix time stamp, meaning the number of seconds since January 1, 1970. Learn more here: https://en.wikipedia.org/wiki/Unix_time . You will be using this time stamp to create two methods, getDate() and timeDiff().

getDate() will return the date and time associated with the time stamp of the Block in a specific format. For example, the following line of code:

```
System.out.println(Blocks.getBlockByNumber(15049333).getDate());
```

Should print to the console:

```
Thu, 30 June 2022 02:56:51 CDT
```

"Thu" is the first three letters of the day of the week the date corresponds to. "30 June 2022" is the date in the form of the day followed by the month fully printed out followed by the year. "02:56:51" is the time in hours:minutes:seconds. The hours should be in 24 hour time, so 13 should appear in the case of 1 pm. "CDT" refers to our time zone.

timeDiff() will take two Blocks as inputs and print to the console the difference in time between them in hours, minutes, and seconds. The order of the Blocks should not matter. For example, the following line of code:

```
Blocks.timeDiff(Blocks.getBlockByNumber(15049333), Blocks.getBlockByNumber(15049367));
```

Should print to the console:

```
The difference in time between Block 15049333 and Block 15049367 is 0 hours, 7 minutes, and 2 seconds.
```

If the inputs are switched, the only thing that will change is the order the Block numbers are listed in.

If the hours, minutes, or seconds equal 1, make sure to print it as hour, minute, and second:

```
The difference in time between Block 1 and Block 2 is 1 hour, 1 minute, and 1 second.
```

If one of the given Blocks objects is null, the output should be:

```
A given Block is null.
```

This method can be done manually through manipulation of the timestamps.

2.4. Transaction Difference:

After that, you will need to implement the transactionDiff() method. This method takes two Blocks as arguments and returns an int which is the number of transactions that occur between those two Blocks (not inclusive). If the Blocks are the same Block, or if the first Block comes after the second Block, or if either of the Blocks are null, you should return -1. For example, the following code example:

```
System.out.println(Blocks.transactionDiff(Blocks.getBlockByNumber(15049322), Blocks.getBlockByNumber(15049325)));
System.out.println(Blocks.transactionDiff(Blocks.getBlockByNumber(15049322), Blocks.getBlockByNumber(15049324)));
System.out.println(Blocks.transactionDiff(Blocks.getBlockByNumber(15049322), Blocks.getBlockByNumber(15049323)));
System.out.println(Blocks.transactionDiff(Blocks.getBlockByNumber(15049322), Blocks.getBlockByNumber(15049322)));
System.out.println(Blocks.transactionDiff(Blocks.getBlockByNumber(15049322), Blocks.getBlockByNumber(15049321)));
```

would return:

353 // adding the transactions from Blocks 15049323 and 15049324
40 // adding the transactions from Block 15049323
0 // no transactions are between consecutive Blocks
-1 // can't count transactions between the same Blocks
-1 // can't count transactions if the second Block is before the first Block

To make sure your transactions are accurate you will first need to sort the blocks ArrayList. This will be done in the sortBlocksByNumber() method. In this method you will sort the blocks ArrayList in ascending order based on the Block number of each Blocks object. To accomplish this, you can implement the comparable interface and override the compareTo method.

3. Unit Tests and Getters:

There are some JUnit tests provided to help you out. Make sure to implement the following getters in the same manner as the tests to make sure Zylabs grades correctly:

- getNumber() should return the Block number
- getMiner() should return the miner address
- getTransactions() should return the transaction_count
- getBlocks() should return a copy of the ArrayList created from reading the file

4. Submission Instructions:

This project requires the submission of *soft copy on ZyLab* and *Github.*

Plagiarism will not be tolerated under any circumstances. Participating students will be penalized depending on the degree of plagiarism. It includes "No-code" sharing among the students. It can lead to academic misconduct reporting to the authority if identical code is found among the students.

Submission Components	Points
Zylabs Submission	80
Github Commits (at least 10)	10
Javadocs	10

5. Late Penalty:

Submit your project before the due date/time. **No late submissions allowed.**

Good Luck!!