CS 2334: Programming Structures and Abstraction
Project1
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.

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.

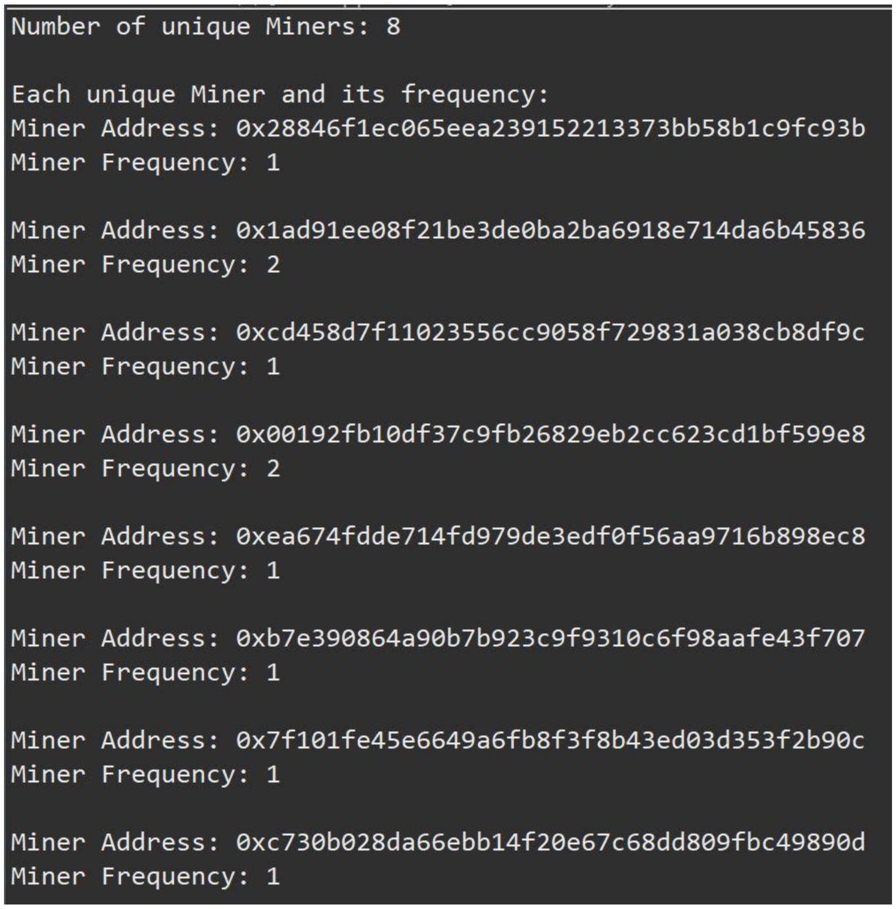
We provided Driver.java and ethereumP1data.txt [do not edit these two files]. You are required to create and write the necessary class, methods, etc. to complete the project based on this project description and Driver.java we provided.

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 block number and miner, columns 1 and 10 respectively in the data 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 as each of these fields will be used later in the project.

2.3. Calculate Unique Miners:

Next you will need to use the data you read in to calculate the number of unique miners and the number of times each of those miners appear in the data file. This will be done from the method call Blocks.calUniqMiners() in Driver.java. The method should output to the console according to this sample output based on the first 10 lines of the data file:



```
Number of unique Miners: 8

Each unique Miner and its frequency:
Miner Address: 0x28846f1ec065eea239152213373bb58b1c9fc93b
Miner Frequency: 1

Miner Address: 0x1ad91ee08f21be3de0ba2ba6918e714da6b45836
Miner Frequency: 2

Miner Address: 0xcd458d7f11023556cc9058f729831a038cb8df9c
Miner Frequency: 1

Miner Address: 0x00192fb10df37c9fb26829eb2cc623cd1bf599e8
Miner Frequency: 2

Miner Address: 0xea674fdde714fd979de3edf0f56aa9716b898ec8
Miner Frequency: 1

Miner Address: 0xb7e390864a90b7b923c9f9310c6f98aafe43f707
Miner Frequency: 1

Miner Address: 0x7f101fe45e6649a6fb8f3f8b43ed03d353f2b90c
Miner Frequency: 1

Miner Address: 0xc730b028da66ebb14f20e67c68dd809fbc49890d
Miner Frequency: 1
```

The first line: "Number of unique Miners: 8" means that there were 8 different miner addresses within the first 10 lines of the data file. Under "Each unique Miner and its frequency:" you will print out the miner address and the frequency it appeared in order of appearance in the data file. For example, "Miner Address: 0x28846f1ec065eea239152213373bb58b1c9fc93b" refers to the miner address of the first Block that was read in from the data file. "Miner Frequency: 1" means that specific address only appeared once in the data. A couple ways to check this method are to make sure the number of unique miners matches the number of entries under "Each unique Miner and its frequency:", and make sure the sum of all the frequencies adds up to the number of Blocks you read in.

2.4. Block Difference:

After that, you will need to implement the blockDiff() method. This method takes two Blocks as arguments and returns an int which is the difference of the two based on their respective Block number. The output can be positive or negative based on the order the Blocks are given in. For example, the following code:

```java
System.out.println(Blocks.blockDiff(Blocks.getBlockByNumber(15049308), Blocks.getBlockByNumber(15049316)));
System.out.println(Blocks.blockDiff(Blocks.getBlockByNumber(15049316), Blocks.getBlockByNumber(15049308)));
```

would return:
-8
8

To supply blockDiff with two Blocks, you will need to create the method getBlockByNumber which takes an int representing the Block number as an argument and returns the Blocks object you read in that matches the supplied number. For example, in the same input above, “15049308” is the Block number corresponding to the Block in the first line of the data file. So, the method should return the Blocks object that was formed from reading that line of data. If the Block number does not refer to an existing Block, you should return null.

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
- 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 to avoid any late penalty. **A late penalty of 20*% per hour* will be imposed after the *due date/time***. After five hours from the due date/time, submissions will not be graded.

Good Luck!!