CMPE 327 Assignment 4

CMPE 327
Assignment 4
November 15th, 2017
Jonathan Turcotte
Omar Sandarusi

Execution

The Back Office can be started from wherever the py file structure is installed. Open a terminal in that directory and run the program using the following command:

```
python backend.py <merged transaction file> <master accounts file>
<new master accounts file> <valid accounts file>
```

The four arguments are all absolute or relative paths to the desired files. The first two are the current locations of the merged transaction file from the previous day's front-end sessions and the previous master accounts file, acting as the two input files. The second two are output file destinations for the new master accounts file that will be generated and the associated valid accounts file for use with the front-end on the next day.

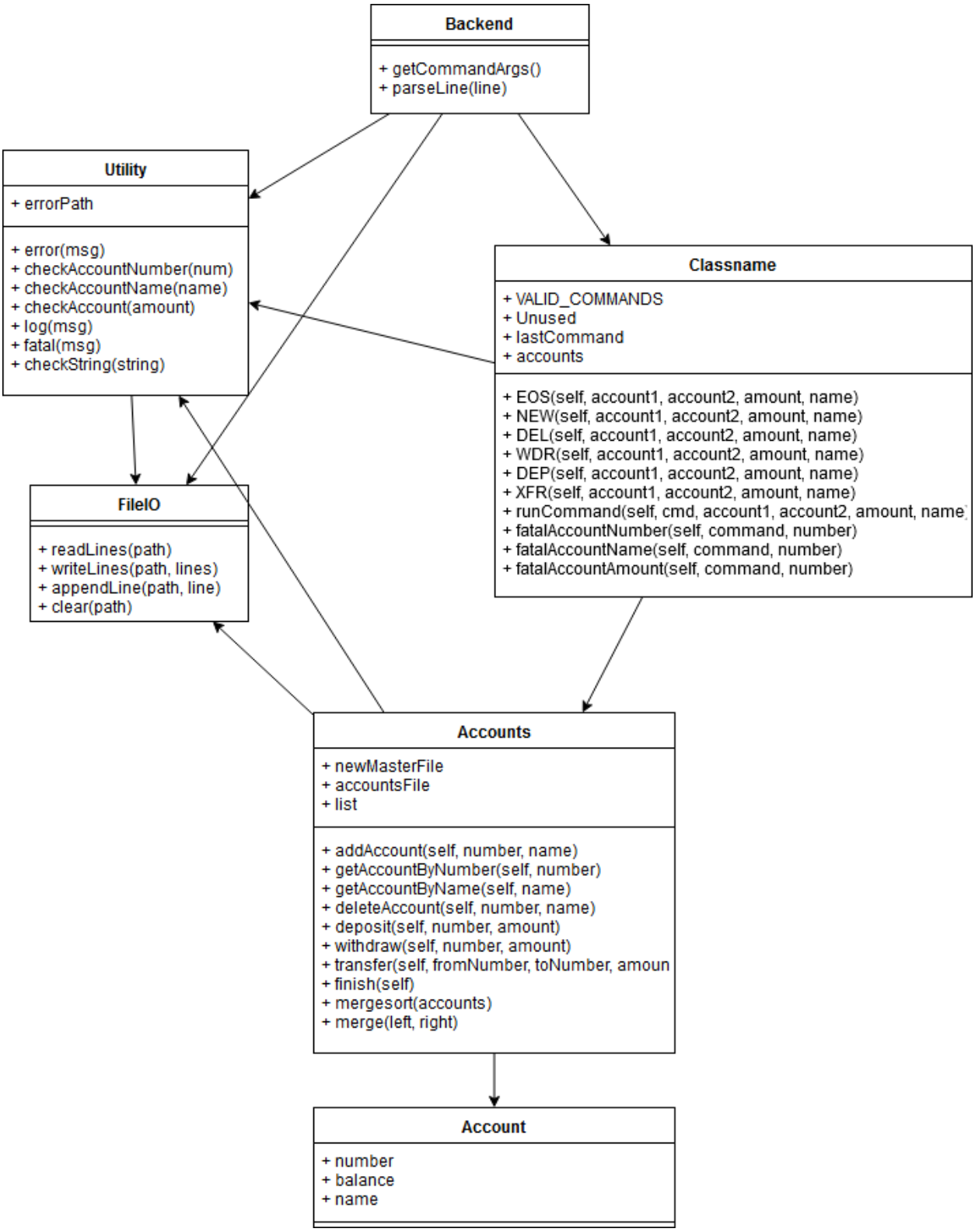
The input files need to exist, and the program will display an error if they cannot be found.

The output files don't have to exist before running, and upon completion the Back-Office will overwrite any file at their locations with the new output files.

Any errors, fatal or not, will be logged to an error.txt file in the working directory of the program.

Design

Our design diagram can be seen below.



Each class is separated into a separate file named after itself. The files must all exist in the same folder to work.

Backend

The entry point into our program, it contains the main loop for iterating through the list of transactions. It also contains functions for parsing command line arguments, instantiating the necessary objects, and parsing the lines of the merged transaction file.

Commands

The `Commands` class is instantiated by the main class, and contains the functions related to each given command. It contains an enumerated list of valid command types, and instantiates the `Accounts` object. The `Commands` class contains functions for the NEW, DEL, DEP, WDR, and XFR commands, which perform all the error checking and validation of inputs before calling the appropriate function for that command in the `Accounts` object. To simplify function calling, the main loop in `Backend` first sends the command to the `runCommand` dispatcher. This dispatcher then calls the appropriate function within `Commands`.

Account

The `Account` class is a simple object that represents a single account in the system. It contains the number, name, and balance of the account, and is completely mutable.

Accounts

The `Accounts` class is responsible for holding the current state of the accounts list and all account-related tasks. On initialization it reads the old master accounts file and parses it into an array of `Account` objects. It exposes functions to perform operations on the account list, like adding an account, depositing, transferring, deleting, etc. It also exposes the finish function, which creates the valid accounts file and new master accounts file based on the current state of the `Accounts` object.

FileIO

`FileIO` is a static class containing useful functions for reading and writing files to and from the disk. It also contains a function for clearing files.

Utility

The `Utility` class is another static class that contains common functions that are used by the other classes. In this case, the utility class contains functions for validating account numbers, names, balances, etc.. as well as displaying and logging errors. This includes functions like `error`, `checkAccountName`, `checkAccountNumber`, and `fatal`.