=============

DOCUMENTATION

=============

================

I - Introduction

================

This milestone incorporates two final changes. First, a forgetfulness mechanism
was implemented, causing peers to be forgotten after a certain period of time, which
can be specified through the command line. Second, a LEAVE command was implemented,
which causes the server to remove the peer whose name is enclosed in the LEAVE command from
the data file storing the list of peers known by the server, effectively forgetting that
peer.

=====================

II - Technologies Used

=====================

The project is implemented in Java, and relies on the Apache Commons CLI library
for parsing command line options. We opted to use the file system rather than a
database library for saving information.

We introduced open source code to produce the hash function for gossip messages
in the client program. Specifically, the SHA256 class was found on github at the
following link:

https://github.com/BaconRemovalUnit/SHA256/blob/master/src/SHA256.java

We also used the provided ASN.1 library, DirectDemocracyP2P, for using ASN.1 in Java found on the class website. This library is on gtihub at the following link: https://github.com/ddp2p/DDP2P/tree/master/src/java/net/ddp2p/ASN1

Generating the user manual for the project will require javadoc. The specific script for doing so is "generateDocs.sh"

==================

III - Architecture

==================

i) Structure

The structure of the project has not changed since the last milestone. The important thing to remember is that all non-networking functionality is defined in the abstract class 'GossipServer', which is where the forgetfulness mechanism was implemented. The only new addition to the structure is an additional datatype class, Leave.java. This class encapsulates the ASN.1 encoding and decoding of the LEAVE command and is used by both the client and the server.

ii) Design Choices

After reflecting on the problem, it became clear that implementing the forgetfulness mechanism is a problem which can be approached in a number of ways. Because we are using a singular file to hold the peers rather than a database, we didn't want constant reading from and rewriting to this file to became a bottleneck during high traffic. However, another simple method for keeping information up to date, simply keeping the set of peers in memory, was impractical because it puts the system at risk for loss of data.

We have implemented a lazy approach which rewrites the peers file when the fraction of peers which are outdated exceeds a certain threshold, while never passing outdated peers to any of the functions which request the set of all peers. This provides the illusion that the file is constantly being trimmed, while avoiding unnecessary file rewrites.

One thing to note about the LEAVE command is that it does cause the server to immediately write to the peers data file to remove the given peer to forget from the data file, unlike the forgetfulness mechanism. The other approach would have involved artificially changing the "last seen" date for the given peer stored in the peers data file to be farther in the past than the period of time provided via command line (or a default of two days) for a peer to be forgotten after. This option seemed more error prone and possibly even a security risk (if the system was used in a real-world scenario) as a peers data would remain in the server's data files even after the server was told to forget data about the peer. As a result, the LEAVE command causes the server to immediately write to the file and forget the peer.

iii) Data File Format Documentation

To track when a peer was last recorded, we needed to add an additional entry into the peers file: the date the peer was last seen. Otherwise, the file system remains unchanged.

gosssip.txt:

Because the gossip messages are forwarded to known peers in the exact format that they are received in, there is no need to change the way that the data is arranged for storage in the gossip data file. Each unique gossip message received is stored in the gossip data file with the format:

```
        <encoding>:<dateTime>:<message>%
```

The first field, GOSSIP, does not need to be included since all data in the gossip data file is already known to be about gossip messages.

peersFile.txt:

Each peer only requires one line since the newline character cannot appear in the information about a peer, so the format is simple. Each line in the peers data file contains information about a single peer in the format:

```
        <name> <port> <ip> <date>  <- date is new for this milestone
```

The four fields can be separated by any amount of whitespace, but must always be separated by at least some whitespace so that they can be read properly.

================

IV - User Manual

================

The user manual can be generated through javadoc by using the script

"generateDocs.sh" I highly recommend making auto-generated user manuals
a requirement in future classes, as it leads to more easily readable
and accessible code. It is a very useful skill to have in general. Javadoc
is not available for C++ users, but Doxygen does basically the same thing
and is language-agnostic.

===============

V - Conclusion

===============

A lazy forgetfulness scheme has been implemented which avoids unnecessary
rewrites to the filesystem. Good object oriented design decisions made earlier on
resulted in this being a relatively simple task. The LEAVE command

================

VI - References

================

This project uses command line option parsing code adapted from Dmitry
Kulakov's example provided during his in-class presentation. The home page of
the parsing code is: http://commons.apache.org/proper/commons-cli/

DirectDemocracyP2P ASN.1 Java Library:
https://github.com/ddp2p/DDP2P/tree/master/src/java/net/ddp2p/ASN1