Assignment 1

Team

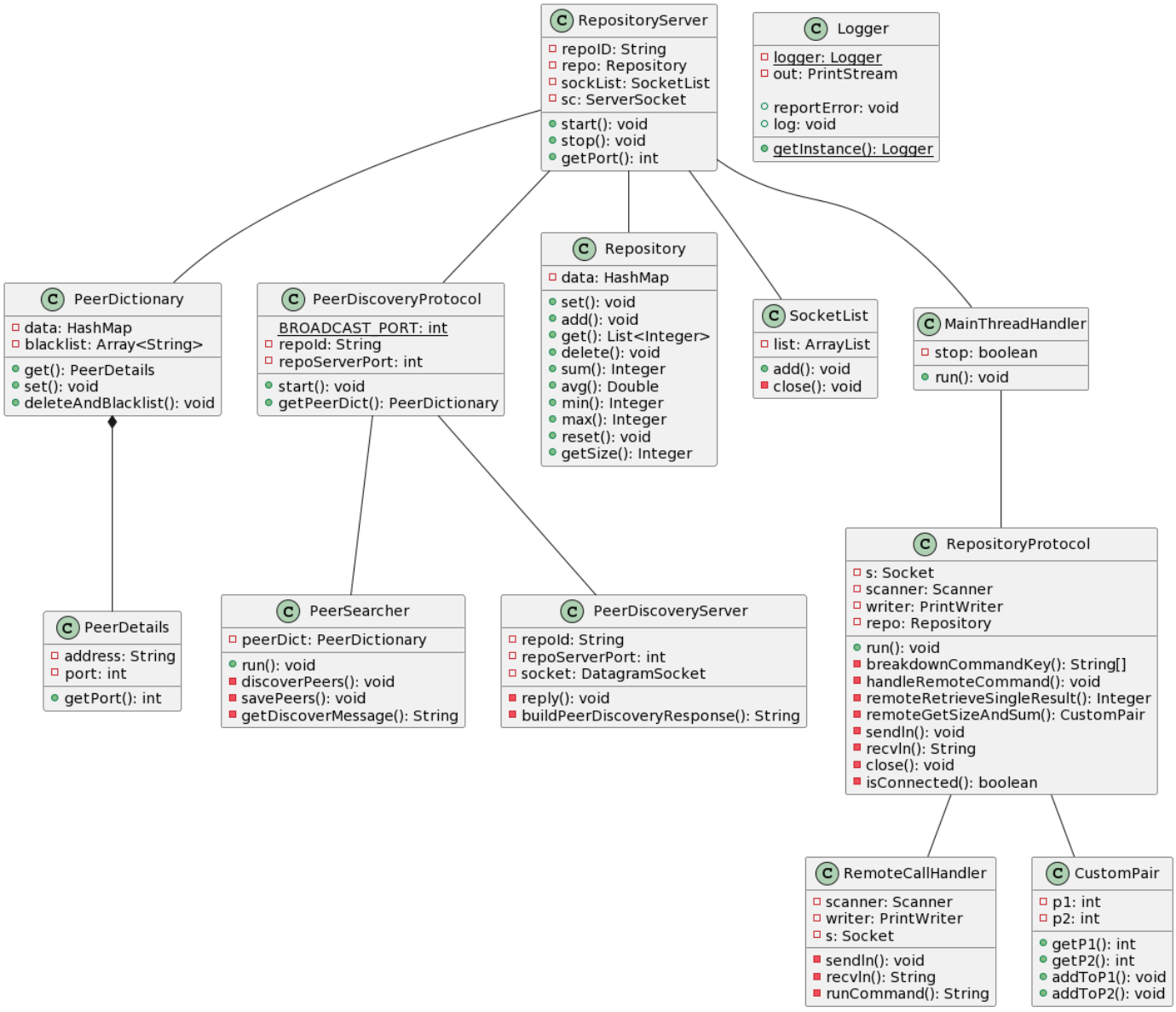
James Juan Whei Tan - 40161156

Vithushon Maheswaran - 27052715

VCS

Development of the project was carried out in a Github repository.

Class Diagram



Repository Core

A repository allows the storage of values (**integers only** for the sake of simplifying things) corresponding to keys (any string is a valid key as long as it does not contain a dot).

Set

Make a particular key correspond to a list containing just the given value.

Add

Adds a value to the list of values that correspond to a particular key. If the key does not exist yet, then a new list will be created for it.

Get

Returns a particular list of values corresponding to a particular key, this returns an empty list if the key does not exist.

Sum

Returns the sum of values corresponding to a particular key. This returns 0 if the key does not exist.

Delete

Delete a particular key-value pair by specifying the key. This is a no-op if the key does not exist.

Min

Returns the minimum value corresponding to a particular key. This returns `null` if the key does not exist.

Max

Returns the maximum value corresponding to a particular key. This returns `null` if the key does not exist.

Avg

Returns the average value (in floating point) corresponding to a particular key. This returns 0.0 if the key does not exist.

Reset

Wipe out the entire repository

Repository Server

A TCP listener that implements the client side (RAP and ERAP) protocol.

Repository Access Protocol (RAP)

All commands are case sensitive. When the server encounters an invalid syntax or an invalid argument, it returns an error message that begins with `ERR`.

Initial connection

When you first connect to a repository via the RAP, you will be greeted with the message `OK Repository <<ID>> ready`. The server identifies itself with a unique ID.

Setting a value

In order to set a value in the repository, the following command should be used.

```
SET <key> <value>
```

This corresponds to the set operation of the repository.

The server responds with the below upon a successful set.

```
OK
```

Adding a value

In order to add a value to a particular key, the following command should be used.

```
ADD <key> <value>
```

This corresponds to the add operation of the repository.

The server responds with the below upon a successful addition.

```
OK
```

Deleting a key

In order to delete a particular key, the following command should be used.

```
DELETE <key>
```

This corresponds to the delete operation of the repository.

The server responds with the below regardless of whether the key exists.

```
OK
```

Getting a key

In order to get the values that correspond to a particular key, the following command should be used.

```
GET <key>
```

This corresponds to the get operation of the repository.

The server responds with the below if the key exists.

```
OK <value>
```

If the key does not exist, the server returns the following

```
OK
```

Summing up the values of a key

In order to sum up the values that correspond to a particular key, the following command should be used.

```
SUM <key>
```

This corresponds to the get operation of the repository.

The server responds with the below upon a successful summation.

```
OK <sum>
```

Finding the minimum/maximum value of a key

In order to find the min/max value that corresponds to a particular key, the following commands should be used.

```
MIN <key>
```

```
MAX <key>
```

This corresponds to the min and max operations of the repository.

The server responds with the below upon a successful aggregation.

```
OK <result>
```

In the event that the key has not been assigned to a value, the following response should be expected instead.

```
OK
```

Resetting a repository

This removes all the data on a particular repository, the command is

```
RESET
```

Upon a successful reset, the server responds with the following

```
OK
```

Peer Discovery

The peer discovery protocol (PDP) makes use of a known address and port for communication, i.e. `230.0.0.0:6789`.

Every 5 seconds, each peer sends out a **discovery message** to the above multicast address. The message has the format

```
DISCOVER <peer-id>
```

e.g. the peer with ID `R1` would send out the message `DISCOVER R1`.

The peers who receive this discovery message should then respond with an **existence message** of the following format to announce their existence

```
ALIVE <peer-id> <address> <port-number>
```

e.g. suppose that the peer `R2` responds to `R1`, it would send the message

```
ALIVE R2 127.123.8.1 60521
```

A peer that receives an **existence message** will parse it and store the details of its fellow peer in a `PeerDictionary`. If 2 different peers with different addresses and ports claim to have the same ID, that ID will then be blacklisted and users will no longer be able to connect to it.

Extended Repository Access Protocol (ERAP)

List peers

In order to find out which peers are connected to a particular server, the client may use the command

```
GET-PEERS
```

After which the server responds with a list of IDs that correspond to its peers.

```
OK R2, R4, R5, R1
```

Remote operations

All the operations that are valid in the RAP are also valid here, users are just given the possibility of specifying which repository should each action be carried out on. For example, the below command would set the key `A` to a value of `10`.

```
SET A 10
```

If the client intends to set the value of `A` on a different repository, he may do so by specifying the exact repository using the dot notation, e.g.

```
SET R1.A 10
```

The above is also valid for all the commands that have been defined in the RAP. Below is a non-exhaustive list of examples

```
GET R5.b
SUM R4.c
ADD R2.d 5
DELETE R1.a
```

When a particular server is requested to carry out an operation on a peer, it forwards the operation to its peer. For instance, if the server `R1` receives the below request

```
DELETE R2.b
```

It searches for the address and port of `R2` in its `PeerDictionary` and sends the following command to `R2`

```
DELETE b
```

`R1` would then forward the response of `R2` to the client.

If the server does not recognise the peer server, e.g. if a user intends to communicate with a server that does not exist or if the server has not been discovered yet via the PDP, the server responds with the below error message

```
ERR Non-existence or ambiguous repository R2
```

Distributed operations

Clients are also able to carry out distributed aggregations

Distributed sum

Suppose that the client is connected to `R1` and wants to sum up the values that correspond to the key `a` on the servers `R1`, `R2`, and `R3`. The below command should be used

```
DSUM a INCLUDING R2 R3
```

This returns the below on a successful summation

```
OK <sum>
```

This does not return an error if the key does not exist on any of the servers, if the key is absent on all the servers the sum would be 0.

Distributed average

Suppose that the client is connected to `R1` and wants to find the average value of `a` across the repositories `R1`, `R2`, and `R3`. The below command should be used

```
DAVG a INCLUDING R2 R3
```

This returns the below on a successful aggregation

```
OK <avg>
```

This does not return an error if the key does not exist on any of the servers, if the key is absent on all the servers the result would be `0.0`, i.e.

```
OK 0.0
```

Distributed min/max

Suppose that the client is connected to `R1` and wants to find the min/max value of `a` across the repositories `R1`, `R2`, and `R3`. The below command should be used

```
DMIN a INCLUDING R2 R3
```

or

```
DMAX a INCLUDING R2 R3
```

This returns the below on a successful aggregation

```
OK <res>
```

This does not return an error if the key does not exist on any of the servers, if the key is absent on all the servers the result would be `null`, i.e.

```
OK
```

Sample run

Launching the server gives out the following output

```
Servers are listening on ports [50494, 50495, 50496, 50497, 50498].

Press hit ENTER if you wish to stop the servers. Note that the service may NOT stop
immediately.
```

We can see the ports on which each repository is listening to. Suppose that we connect to R1 (the first port) and run the following commands, we can expect the following behaviour

```
OK Repository <<R1>> ready
>>> SET a 12
OK
>>> DELETE b
OK
>>> ADD a 13
OK
>>> GET a
OK 12, 13
>>> SUM a
OK 25
>>> AVG a
OK 12,50
>>> MIN a
OK 12
>>> MAX a
OK 13
>>> SET a 10
OK
>>> GET a
OK 10
>>> RESET a
OK
>>> GET a
OK
>>> SET a 5
OK
>>> ADD a 2
OK
>>> SET R2.a 10
OK
```

```
>>> GET a
OK 5, 2
>>> GET R1.a
OK 5, 2
>>> GET R2.a
OK 10
>>> SET R3.a 20
OK
>>> DSUM a INCLUDING R2 R3
OK 37
>>> DMAX a INCLUDING R2 R3
OK 20
>>> DMIN a INCLUDING R2 R3
OK 2
>>> DAVG a INCLUDING R2 R3
OK 9,25
```