Assignment 2 Report

The Following tests and resources can be found in src/ReportAndTesting/UnitTesting.java

Unit Tests

These tests are to verify the basic functionality of the core components of the blockchain server. More advanced tests are run in the Integration Test section, while these are relatively basic.

Blockchain functionality tests

```
//******************TEST BLOCKCHAIN FUNCTIONALITY*****************//
//add one transaction
public void addOneTransaction()

Purpose: test general adding of transaction to blockchain
Input: tx|test1111|1
Expected Output: successfully added value to blockchain
Actual Output: true if blockchain pool is 1, and successfully added.


//add multiple transaction
public void addMultipleTransaction()

Purpose: test general adding of multiple transaction to blockchain
Input: tx|test1111|1, tx|test2222|1, tx|test3333|1
Expected Output: successfully added all values to blockchain
Actual Output: true if blockchain pool is 3, and all successfully added.

//add invalid transactions
public void addInvalidTransaction()

Purpose: test adding multiple transactions, including invalid values
Input: tx|test|1 (invalid), txx|test1234|1 (invalid), tx|test12345|1 (invalid)
Expected Output: none of the inputs are added to the chain
Actual Output: true if blockchain pool is 0, and none successfully added added.
```

Server Info Functionality

```
//*********************TEST SERVERINFO FUNCTIONALITY********************//
//create & retrieve one
public void createServerInfo()

Purpose: test general creation of serverInfo object with valid inputs
Input: host: “localhost”, port: 8333
Expected Output: successfully created serverInfo
Actual Output: true if able to access host and port values from object

//Create invalid server infos (should return null for invalid host, and 0 for
invalid port).
public void createInvalidServerInfo()
Purpose: test creation of serverInfo objects with invalid inputs
Input: host: “”, port: 8333
Input: host: “localhost”, port: 900
```

Expected Output: created infos, but returns null or 0 when accessing.
Actual Output: true if null or 0 is returned for invalid entry

Server Info List

```
//*********************TEST SERVERINFOLIST FUNCTIONALITY**********************//
//Initialize from file1, a simple config file, check all values match provided
             /*     servers.num=1
                    server0.host=localhost
                    server0.port=8334   */
public void configSimple()
```

Purpose: test initialisation from file works on basic test
Input: file1.txt (as above)
Expected Output: read correctly, created list with 1 server
Actual Output: true if size is correct, and values can be accessed

```
//Initialize from file2, a simple config file with multiple servers and spacing,
check all values match provided
      /*     servers.num=3

             server0.host=localhost
             server0.port=8334

             server1.host=globalhost
             server1.port=8333

             server2.host=127.22.0.420
             server2.port=8335   */
public void configMultipleSimple()
```

Purpose: test initialisation from file with multiple servers works on basic test
Input: file2.txt (as above)
Expected Output: read correctly, created list with 3 servers
Actual Output: true if size is correct, and values can be accessed

```
//Initialize from file3, a config file with multiple servers, including
inconsistent pairing, invalid ports, and too many servers.
      /*     servers.num=3

             server0.host=localhost

             server1.host=globalhost
             server1.port=123456

             server2.host=localhost
             server2.port=8335

             server3.host=localhost
             server3.port=8333   */
public void configComplexOne()
```

Purpose: test initialisation from file works with corner cases & inconsistencies
Input: file3.txt (as above)
Expected Output: read correctly, created list with 3 servers, null where invalid

```
Actual Output: true if size is correct, and values can be accessed or are null


//Initialize from file4, a config file with multiple servers, including
inconsistent pairing, empty values, and too many servers.
      /*     servers.num=5

             server0.host=localhost
             server1.host=globalhost
             server1.port=123456

             server2.host=localhost
             server2.port=8335
             server3.host=localhost
             server3.port=8333

             server0.host=mega
             server5=
             server5=
             server4.host=megaman
             server4.port=6666
                    */
public void configComplexTwo()

Purpose: test initialisation from file with several problems
Input: file4.txt (as above)
Expected Output: read correctly, created list with 5 servers (null where invalid)
Actual Output: true if size is correct, and values can be accessed

//Initialize from an invalid filepath, to check correct error is thrown (file not
found)
public void configInvalid()
Purpose: test if invalid filepaths are handled correctly
Input: invalid path
Expected Output: “File not found.”
Actual Output: true if throws FileNotFoundError with correct message
```

Integration Testing

These tests are to test the functionality of services intercommunicating. They are run on the two core components of the task, Server and Client.

Server Tests

```
//adds 1 valid and 1 invalid transaction, checks correct output returned
public void addTransactionTest1()
Purpose: test that single transaction can be added & rejected correctly
Input: valid and invalid transaction
Expected Output: returns accepted for one rejected for the other.
Actual Output: true as the above is correct

//adds 4 valid transactions, 2 invalid, checks correct output returned
public void addTransactionTest2()
Purpose: test adding multiple transactions, both valid and invalid
Input: valid, invalid, valid, invalid
Expected Output: accepted, rejected, accepted, rejected
Actual Output: true as the above is correct
```

```
//adds 2 valid transactions, calls pb, checks correct output returned
public void printTransactionTest1()
Purpose: add two transactions and then test that pb output is correct
Input: 2 valid inputs
Expected Output: correct block printing format
Actual Output: correct, true if matches expected format

//adds 4 transactions (3 valid), calls pb, checks correct output returned
public void printTransactionTest2()
Purpose: add both valid and invalid transactions, verify pb is correct
Input: valid, valid, valid, invalid
Expected Output:correctly printed chain with 3 of the 4 inputs in pool
Actual Output: true as output matches expected

//input an invalid string, checks correct output returned (unknown command)
public void unknownCommand()
Purpose: verify unknown command is printed if unexpected input
Input:any invalid string
Expected Output: “Unknown command”
Actual Output: “Unknown command”
```

Client Tests

```
//Initialises a client server from  test file 1, then calls List
public void listTest1()
Purpose: test that initialised values are listed correctly
Input: ls
Expected Output: list of servers reflecting initialisation file
Actual Output: as expected

//Initialises a client server from  test file 2 with multiple servers, then calls
List
public void listTest2()
Purpose: test that multiple initialised values are listed correctly
Input: ls
Expected Output: list of servers reflecting initialisation file
Actual Output: as expected

//Initialises a client server from  test file 1, then updates the value
public void updateTest1()
Purpose: test that values are updated correctly
Input: up|0|hostname|port, ls
Expected Output: prints the updated server value
Actual Output: as expected

//Initialises a client server from  test file 1, then attempt to update with
invalid value
public void updateTest2()
Purpose: test that values are not updated incorrectly
Input: up|0|hostname|invalidPort, ls
Expected Output: prints the original server value
Actual Output: as expected

//Initialises a client server from  test file 1, then attempt to update with no
value provided
public void updateTest3()
Purpose: test that values are not updated incorrectly
Input: up|0|, ls
Expected Output: prints the original server value
Actual Output: as expected
```

```
//Initialises a client server from  test file 5, which creates server 1 and 2,
then call clear
public void clearTest()
Purpose: clears all null values from the list
Input: cl
Expected Output: all values in list are shifted down to fill null spots
Actual Output: as expected, all indexes shift down

//Initialises a client server from  test file 5, which creates server 1 and 2,
then removes server at index 1
public void removeTest()
Purpose: remove a server via index
Input:rm|index
Expected Output: Succeeded, server details
Actual Output: as expected

//Initialises a client server from  test file 5, which creates server 1 and 2,
then removes an invalid index
public void removeInvalidTest()
Purpose: remove a server via invalid index to test error handling
Input:rm|indexOutOfBounds
Expected Output: Failed, original server details
Actual Output: as expected
```

Acceptance Testing

The following test cases were run by providing non-students (i.e. family) with a list of basic commands, without specifying error or requirements. The following are their provided config files, and the result of the test.

Test Case 1:

```
Servers.num=4

Server0host=firsthost

Server0port=0

Server1host=

Server1port=1

Server2host=myhost

Server2port=9000

Server4host

Server4host=1000
```

Result: No unexpected errors encountered, code handled error correctly. Produced a server list of size 4, consisting of all null values due to the formatting error (missing . between Server[num] and host or port.

Test Case 2:

```
Servers.num=10

Servers.num=0

Server0.host=test

Server0port=7000

Server1.host=test2

Server1.port=7000

Server2host=test

Server2.port=9000
```

Result: No unexpected errors encountered, code handled error correctly. Produced a serverInfo list of size 0.