95-702 Distributed Systems For Information Systems Management

Project 3

Assigned: Friday, February 22, 2019
Due: Friday, Due Friday March 8, 11:59pm

Web Service Design Styles

Principles

One of our primary objectives in this course is to make clear the fundamental distinction between functional and nonfunctional characteristics of distributed systems. The functional characteristics describe the business or organizational purpose of the system. The non-functional characteristics affect the quality of the system. Is it fast? Does it easily interoperate with others? Is it fault tolerant? Is it reliable and secure?

In this project, we illustrate several important nonfunctional characteristics of distributed systems. We will use web services to enhance interoperability. We will consider various styles of API design. We will build a stand-alone blockchain and remote clients that interact with a blockchain API.

Overview

In Task 0, you will write a blockchain by carefully following the directions in Javadoc format found here:

http://www.andrew.cmu.edu/course/95-702/examples/javadoc/index.html

In the remaining tasks, we will provide three different interfaces to this blochchain implementation.

In Task 1, you will deploy some of the blockchain methods as a SOAP based API. You will visit this API with a SOAP client (transmitting simple transactions). The blockchain will hold the transactions. There is a video here that will help in building a JAX-WS RPC-Style web service.

http://www.andrew.cmu.edu/course/95-702/video/SimpleJAX-WS/index.html

In Task 2, you will deploy the blockchain methods using a single argument style JAX-WS web service. See the videos on the course schedule titled “Service Design Styles” one through six. This still uses SOAP but has a different design style.

In Task 3, you will deploy the blockchain methods using a resource based or REST web service. As before, the blockchain will be used to store simple transactions. See the videos on the course schedule titled “Service Design Styles” one through six.

Before beginning, be sure to study the Javadoc on the schedule. It describes the two classes that you need to write – Block.java and BlockChain.java.

For each task below, you must submit screenshots that demonstrate your programs running. These screenshots will aid the grader in evaluating your project.

Documenting code is also important. Be sure to provide comments in your code explaining what the code is doing and why. Be sure to separate concerns when appropriate. You may include the Javadoc comments (provided) in your own code. But do comment on any additions or modifications that you make.

Task 0 Execution

Write a solution to Task 0 by studying the Javadoc provided on the course schedule. You will need to make modifications to the code for the other tasks. Task 0 is essential. The logic found in Task 0 will be reused in Tasks 1, 2, and 3.

The execution of Task 0, a non-distributed stand-alone program, will look almost exactly like the following interaction. You will select the exact same options and enter the exact same transactions. The only differences, of course, will be those due to the dynamic computations associated with the blockchain. As part of the submission of Task 0, you must turn in a screen shot such as the following:

In Netbeans, select
run:
run:
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by repairing the chain.
6. Exit
0
Current size of chain: 1
Current hashes per second by this machine: 1846198
Difficulty of most recent block: 2
Nonce for most recent block: 1154

Chain hash:
00BC4767DC821A3F4B64B42017228734251FF1C0FF70EDDF9A66DC2C1AC7EFD8
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
2
Verifying entire chain
Chain verification: true
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
3
View the Blockchain
{"ds_chain" : [{"index" : 0,"time stamp " : "2019-02-22 17:22:14.133","Tx ":
"Genesis","PrevHash" : "","nonce" : 1154,"difficulty": 2}
 ],
"chainHash":"00BC4767DC821A3F4B64B42017228734251FF1C0FF70EDDF9A66DC
2C1AC7EFD8"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
1
Enter difficulty > 0
5
Enter transaction
Alice pays Bob 100 USD
Total execution time to add this block was 28071 milliseconds

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
1
Enter difficulty > 0
5
Enter transaction
Bob pays Eve 50 USD
Total execution time to add this block was 2066 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
1
Enter difficulty > 0
6
Enter transaction
Eve pays Charlie 10 USD
Total execution time to add this block was 47166 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
2
Verifying entire chain
Chain verification: true
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.

3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
3
View the Blockchain
{"ds_chain" : [{"index" : 0,"time stamp " : "2019-02-22 17:22:14.133","Tx ": "Genesis","PrevHash" : "","nonce" : 1154,"difficulty": 2},
{"index" : 1,"time stamp " : "2019-02-22 17:27:21.442","Tx ": "Alice pays Bob 100 USD","PrevHash" : "00BC4767DC821A3F4B64B42017228734251FF1C0FF70EDDF9A66DC2C1AC7EFD8","nonce" : 1457348,"difficulty": 5},
{"index" : 2,"time stamp " : "2019-02-22 17:28:27.419","Tx ": "Bob pays Eve 50 USD","PrevHash" : "0000097BCBDF5146BAA13491D52B32946BE9ED8A7F0262EBC47A9AA7213E85F7","nonce" : 107856,"difficulty": 5},
{"index" : 3,"time stamp " : "2019-02-22 17:29:16.839","Tx ": "Eve pays Charlie 10 USD","PrevHash" : "0000017BA1238C419C992654EAFC5417F1BDBAF2FECFCF44328B0CCD777EDBF4","nonce" : 2475995,"difficulty": 6}
],
"chainHash":"000000B2594B8EE49149604B87835990E6EF36C2371768A13A36B2F1EE2EF3EF"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
0
Current size of chain: 4
Current hashes per second by this machine: 1946252
Difficulty of most recent block: 6
Nonce for most recent block: 2475995
Chain hash:
000000B2594B8EE49149604B87835990E6EF36C2371768A13A36B2F1EE2EF3EF
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.

3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
4
Corrupt the Blockchain
Enter block ID of block to Corrupt
2
Enter new data for block 2
Charlie pays Dave 3 USD
Block 2 now holds Charlie pays Dave 3 USD
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
3
View the Blockchain
{"ds_chain" : [{"index" : 0,"time stamp " : "2019-02-22 17:22:14.133","Tx ": "Genesis","PrevHash" : "","nonce" : 1154,"difficulty": 2},
{"index" : 1,"time stamp " : "2019-02-22 17:27:21.442","Tx ": "Alice pays Bob 100 USD","PrevHash" : "00BC4767DC821A3F4B64B42017228734251FF1C0FF70EDDF9A66DC2C1AC7EFD8","nonce" : 1457348,"difficulty": 5},
{"index" : 2,"time stamp " : "2019-02-22 17:28:27.419","Tx ": "Charlie pays Dave 3 USD","PrevHash" : "0000097BCBDF5146BAA13491D52B32946BE9ED8A7F0262EBC47A9AA7213E85F7","nonce" : 107856,"difficulty": 5},
{"index" : 3,"time stamp " : "2019-02-22 17:29:16.839","Tx ": "Eve pays Charlie 10 USD","PrevHash" : "0000017BA1238C419C992654EAFC5417F1BDBAF2FECFCF44328B0CCD777EDBF4","nonce" : 2475995,"difficulty": 6}
],
"chainHash":"000000B2594B8EE49149604B87835990E6EF36C2371768A13A36B2F1EE2EF3EF"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.

3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
2
Verifying entire chain
..Improper hash on node 2 Does not begin with 00000
Chain verification: false
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
5
Repairing the entire chain
Total execution time required to repair the chain was 26132 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
3
View the Blockchain
{"ds_chain" : [{"index" : 0,"time stamp " : "2019-02-22 17:22:14.133","Tx ": "Genesis","PrevHash" : "","nonce" : 1154,"difficulty": 2},
{"index" : 1,"time stamp " : "2019-02-22 17:27:21.442","Tx ": "Alice pays Bob 100 USD","PrevHash" : "00BC4767DC821A3F4B64B42017228734251FF1C0FF70EDDF9A66DC2C1AC7EFD8","nonce" : 1457348,"difficulty": 5},
{"index" : 2,"time stamp " : "2019-02-22 17:28:27.419","Tx ": "Charlie pays Dave 3 USD","PrevHash" : "0000097BCBDF5146BAA13491D52B32946BE9ED8A7F0262EBC47A9AA7213E85F7","nonce" : 441725,"difficulty": 5},
{"index" : 3,"time stamp " : "2019-02-22 17:29:16.839","Tx ": "Eve pays Charlie 10 USD","PrevHash" :

"00000B1C711B1D81B7724D0FC11C3D2AC51345E505B4CF72121D246519C201
01","nonce" : 852425,"difficulty": 6}
 ],
"chainHash":"000000D44CB102756678DBE10C321D2A93F9FCA9A8FE370FF888C
18CAF7C11D0"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
2
Verifying entire chain
Chain verification: true
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the Corruption by recomputing hashes.
6. Exit
6

Task 0 Grading Rubric 25 Points

See the Javadoc for a description of the main routine and the difficulty levels.

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works. A screen scrape is provided that shows the exact same user selections.	19	17	10	0
The main method of the BlockChain class is documented and describes behavior with difficulty levels of 4 and 5 (see Javadoc)	4	3	2	0
Separation of concerns & good style	1	0	0	0
Submission requirements met	1	0	0	0

Task 1 Execution

In Task 1 we will deploy a set of methods from our Task 0 Blockchain.

The execution of Task 1 will look almost exactly the same as the execution of Task 0. However, our users will be remote and we are not allowing remote users to corrupt the blockchain or repair it. Nor will we allow the external user to view the status (option 0). So, we will have only four options (rather than seven) for our remote

users. While the interaction will appear local, the Task 1 client will make SOAP requests to your newly written blockchain API.

Your task is to design and implement the API. You need to come up with your own method names and method signatures. You may use any of the code from Task 0 that you need. Your client needs to make calls to the API so that it has almost the same execution as Task 0 (without the Task 0 options - 0, 4, and 5). Your screen shot will show the user running and testing each option. No example execution is provided here.

Task 1 Grading Rubric 25 Points

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works and the requirements are met.	20	15	10	0
The interactive screen scrape tests each option.	3	2	1	0
Separation of concerns & good style	1	0	0	0
Submission requirements met	1	0	0	0

Task 2 Execution

The execution of Task 2 will look exactly the same as the execution of Task 1. Behind the scenes, this Task differs from Task 1 in that we are using a single message argument style design. The blockchain service will need to be redesigned. See the Service Design Style videos.

Your API on the server side will only provide one operation. That operation will extract the necessary information from its single message input string and decide what internal operation to perform. You are free to choose how you represent data within

the string. You may choose from a JSON message, an XML message, or comma separated values. Any of these textual representations will work fine.

Task 2 Grading Rubric 25 Points

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works and the requirements are met with a single message argument and a single operation.	20	15	10	0
The interactive screen scrape tests each option.	3	2	1	0
Separation of concerns & good style	1	0	0	0
Submission requirements met	1	0	0	0

Task 3 Execution

The execution of Task 3 will look exactly the same as the execution of Task 1. Behind the scenes, however, the Task 3 client will make REST or resource style requests to your blockchain API. This differs from Task 2 in that here we are using a REST style design. The blockchain service will need to be redesigned for REST. See the Service Design Style videos. Also, in order to write the required Java client, see Lab 7 here:

http://www.andrew.cmu.edu/course/95-702/Labs/RESTLab.txt

You need to think about what is actually a resource and what methods are appropriate to interact with it. You will also need to consider the format of the messages that are passed back and forth. Make reasonable decisions.

The client side code will use a proxy design. In other words, all of the client side communications related code will be isolated within proxies. Your client should provide a demonstration of each operation (as in Task 1).

Task 3 Grading Rubric 25 Points

	Excellent	Good – minor problems	Poor – serious problems	No submission
Works as specified. The interactive screen scrape tests each option.	10	8	5	0
REST design is used for client and server	9	3	1	0
Style and separation of concerns and proxy design	5	3	2	
Submission requirements met	1	0	0	0

Notes on Task 2 API Design - Single Message Style with XML

In this task, you will pass a single message (containing a single String holding an XML or CSV or JSON message) to a single JAX-WS service operation. The service will examine the message and perform the required operation.

In this Task, if you choose to design your message format in XML, your client will need to write that message and your server will need to parse it. Use a parser only when

reading XML (in this case, only on the server). There is no requirement to design a return message format. For those operations returning string data, a simple Java string will be fine.

For parsing the XML, you may use Java's parser.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder;
Document sensorMessage = null;
builder = factory.newDocumentBuilder();
blockChainMessage = builder.parse( new InputSource( new StringReader(
                                        xmlString ) ) );
```

Once the xmlString is converted to a Document object, we can read data from the Document object with code like the following:

```
blockChainMessage.getDocumentElement().normalize();
System.out.println("Root element :" +
blockChainMessage.getDocumentElement().getNodeName());
```

This gets you started. There are still issues needing thoughtful work. Make reasonable decisions.

Notes on Task 3 API Design – Resource or REST style

In this style, you will make good use of HTTP verbs, status codes and URL's.

You will not be using JAX-WS or JAX-RS for this task. Instead, use a plain servlet.

The HTTP status codes may be set on the server side by using the setStatus() method of the response object. To learn what representation the client requires, use the getHeader("Accept") method of the request object.

The client side code (standard Java SE) will perform communication using the URL class and the HttpURLConnection class. For example, in order to make a call to the get method, your code might look something like this:

```
URL url = new URL(http://localhost:8080/Project3Task3/BlockChainService");

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestMethod("GET");

con.setRequestProperty("Accept", "text/xml");
```

The client side code can use the con.getResponseCode() method to examine the HTTP response code. It can also use the con.getInputStream() method to read the server's response body.

Hint: You may use StackOverflow for examples on how to use the HttpURLConnection. If you use any code from StackOverflow, be sure to cite it in your comments.

Note on blockchains

This is not all of blockchain. There is more to learn. Real blockchains include peer to peer communication and many replicas of the blockchain. The blocks themselves include Merkle Trees and real transactions are signed and more carefully crafted. This assignment will certainly provide a nice foundation to build on.

Questions

Questions should be posted to Piazza. If there are missing pieces in this assignment write up (we are sure there are) then make reasonable decisions. Try to build a system that is well thought out and keep things as simple as possible.

Project 3 Submission Requirements

Documentation is always required.

Remember to separate concerns.

The Netbeans projects will be named as follows:

- Project3Task0 (You need to zip this folder)
- Project3Task1Server (You need to zip this folder)
- Project3Task1Client (You need to zip this folder)
- Project3Task2Server (You need to zip this folder)
- Project3Task2Client (You need to zip this folder)
- Project3Task3Server (You need to zip this folder)
- Project3Task3Client (You need to zip this folder)

You should also have four screen shot folders:

- Project3Task0-screenshot (Do not zip)
- Project3Task1-screenshot (Do not zip)
- Project3Task2-screenshot (Do not zip)
- Project3Task3-screenshot (Do not zip)

For each Netbeans project, use “File->Export Project->To Zip”. You must export in this way and NOT just zip the Netbeans project folders. In addition, zip all the Netbeans export zips and the screenshot folders into a folder named with your andrew id.

Zip that folder and submit it to Canvas.

The submission should be a single zip file. This file will be called YourAndrewID.zip.

Submission file structure:

- YourAndrewID.zip
  - Project3Task0.zip
  - Project3Task1Server.zip
  - Project3Task1Client.zip
  - Project3Task2Server.zip
  - Project3Task2Client.zip
  - Project3Task3Server.zip
  - Project3Task3Client.zip
  - Project3Task0-screenshot
  - Project3Task1-screenshot
  - Project3Task2-screenshot
  - Project3Task3-screenshot

Note on Handling Jar files

To add a .jar to your project so that it is included when you zip the project:

To ensure that the .jar that you have included under your Libraries folder in the NetBeans Project is also zipped along with your project, follow these steps:
1. From Windows File Explorer, create a folder named lib under your project folder. i.e., if this is you project folder
path, C:\Users\username\Documents\NetBeansProjects\Project1Task2 then this is where you will create the lib folder. (this is an example path, it will change according to your system)
2. Within the lib folder, copy and paste your downloaded .jar files.
i.e., your .jar files go
here: C:\Users\username\Documents\NetBeansProjects\Project1Task2\lib (this is an example path, it will change according to your system)
3. In NetBeans, go to the Libraries under your Project1Task2 and delete the existing .jar file that you had earlier added.

4. Right click Libraries and choose Add JAR/Folder
5. Browse to the lib folder you created in step 1. The .jar you added to the lib folder

(step 2) will be visible there. Click on that .jar Ensure that 'Reference as': is set to the radio button 'Relative Path'
6. Click Open and you're done!
7. Repeat steps 4-6 for any more .jar