# Forum Postings

## Say Hello Postings:

### Say Hello Forum My Entry:

Hello everyone,

       My name is Thomas Leedberg but I've gone by Tommy for just about my whole life. I'm 31 years old and am a transplant from Massachusetts and have I've lived in Chicago for a little over a year and a half now. I received my undergrad in Computer Science at the University of Massachusetts Lowell, but prior to that I was attending the University of West Virginia. I currently work for Schneider Electric and I've been with them for a little over 8 years now. This is only the second course that I've taken at DePaul and so far I'm enjoying the school. In my spare time I like to travel, I've been a lot of places but there are so many more I want to see and experience. I love being a part of cultures that are not my own. I also enjoy being outdoors, growing up in northern Massachusetts there is plenty of opportunity to go hiking and camping in the mountains of New Hampshire. I also love music and going to music festivals/concerts.

### Reply to Chris Williams Introduction:

Hi Chris,  
        How's it going? I live pretty close to uptown, I was in Ravenswood and now I'm over on Irving Park. I'm a pretty big gym advocate myself. I spend a lot of my free time at Uptown Fitness. I've always had some interest in mobile development, I've only looked into it lightly though most of my focus ends up being cloud based. I have done a couple small projects using cordova for work hackathons that I thought was interesting. I do like that you can touch and feel mobile applications in a way you can't really do with other platforms.

JokeServer and Inet  
Just a suggestion for if you're having trouble getting things running correctly or understanding what's going on in the inet server/client code. In IntelliJ you can host both the server and the client in a debug session simultaneously. I always find it's a lot easier to understand something when I can step through it.

### Reply to Chuan Chang in Hello: Hi John,

Congratulations on graduating this quarter! What do you plan to do after you're done?

General Homework Submissions:  
Hello,

   Is there a file name format we should give our zip files when we submit our programs? I was trying to find it in the docs/content but I couldn’t find anything specific.

### Provide automated grading scripts for submissions double check

Eh what production system" It works on my machine" lol.

### Network Labs Secreenshots?

Some of the network lab questions require you to provide screen shots, but we're only supposed to be submitting our answers in the HTML check list which doesnt have images unless we host them somewhere and provide the link for them. So my question is are we still supposed to provide these screenshots and how?

## JokeServer: “Address Already In Use”

You can also download the free windows tool [TCPview](https://docs.microsoft.com/en-us/sysinternals/downloads/tcpview" \t "_blank)to see what ports are in use and kill it that way, that's how I do it when my program doesn't shut down properly so the port remains open.

## Expert Postings:

### Reply in Joke Server to “Maintaining State/Programming Form Question:”

The short answer is what you are proposing will work just fine. If it's a static final object you wont need to pass it around though, just access it. My suggestion would be to use a Dictionary or Map object over an ArrayList if you want to implement it this way though, then you can have the keys be the user id's and the values be your jokes/proverbs. This would make for much faster lookup.  
  
There is one thing to keep in mind. If you do the above and I read your post right you're structure would look something like the following:  
  
[{ "id1": [ JA, JB, JC, JD]},  { "id2": [ JA, JB, JC, JD]} ,  { "id3": [ JA, JB, JC, JD]} ]  
  
This is good for a small set of users, but imagine having 50-100k users. You're now keeping a list of every joke for every user and could end up being a resource issue.  
  
There are a lot of ways around this but if you want to keep a similar structure one possible option would be to create 2 objects rather than 1. One to maintain the jokes and one to maintain the users. The jokes could be a mapping of every joke you have and an ID for that joke and then the users would be a mapping of user ids and a list of joke objects containing the jokes id and if it's been heard yet. It adds a little more overhead in processing for lookup up the jokes and also determining if they've been heard or not but you save a ton of space by not replicating the jokes. Also, if you want to change a joke you only have to do it in 1 location.

Reply in Networks Lab to “Some other network tools to check out”  
I've actually used both PostMan and Fiddler a lot. One thing I like about fiddler is you can script a lot of stuff into it. For my job part of our platform is a REST API that requires a auth0 bearer token in order to allow you to have access. What I ended up doing was changing fiddler so that when a request was made to any url that contained the platforms url it automatically made a secondary request to get Auth0 to get bearer token and then changed the request that was going to our REST api to add the authentication header. It tooks a little figuring out at first because the fiddler scripting language is kinda funky about ordering but it ended up being a major time saver. I've also done something similar in POSTMan by chaining requests together, although I admit I've used it a lot less though.

JokeServer Joke Server ArchitectureI think we're all on the same page about wishing to be able to use multiple files rather than keep everything in one file. Even with folding it's still a really big jumble. It does make me appreciate all those classes way back when I started that tell you to break things out though lol.  
  
For my implementation I also did a pretty thin client and heavy server. My original plan was to do a mix mode implementation where a small amount of data was stored on the server and the rest was stored on the client but some of the requirements for the assignment made me change my mind in that.  
  
Anyway, onto the actual implementation:  
  
Rather than do a separate class implementation for both joke and proverb mode I kept it pretty simple and did an enum that was then used to determine what the server was supposed to serve up to the client. I would have likely followed your decision to do this in separate classes had there been more than just strings being returned though.   
  
I created class for storing Joke and Proverb information, named JokeProverb for lack of better naming ideas. It was a pretty basic data structure that contained a body and an id originally it contained more data than that but i pulled it out. I could have just used another HashMap here for the jokeId/Key as well. 

This data structure was stored in two arrays, one for Jokes and one for Proverbs, it allowed me to keep 1 single copy of every joke and proverb rather than a copy in each user set. Then I created 2 LinkedHashMaps with the following signature LinkedHashMap<String, LinkedHashMap<String, Boolean>>. The outer map had a key for the UUID of the user( in my case i just used the users name ) and a value of a second LinkedHashMap that had a key for the joke or proverbs id and a boolean value indicating if it had been heard or not yet. I had originally thought to include a flag in the JokeProverb class for if it had been heard or not but this is actually a faster lookup and a bit easier to implement.  
NOTE: Something I actually learned in this project was that you need to use a LinkedHashMap if you want order to maintain. I was originally using a HashMap and when i went to randomize the jokes/proverbs they were constantly being reorganized by key which was not helpful.  
  
Communication between the Server and the Client was done through a data structure I created called a ServerRequest. It had members for a UUID and a String request, I had originally thought we would need to send a true request to the server not just a hit return to get the next joke, this way works for that as well but it's just a bit more extensible if I need it for another project later.  
  
I had to build in some basic serialization/deserialization into the class because I was only passing strings back and forth from the client to the server. Typically, i'd have just a 3rd party library for this though.  
  
When a request came in I would call a helper method that then gets the next joke or proverb by looping through the list of proverbs or jokes associated with that user and finding the first key( the id of the joke/proverb) who's value is false, this bit of work was kind of ugly and could probably use some refactoring, once it has the key it will do a lookup in the Jokes/Proverbs array and then return that joke. Finally there was a string builder method that put everything together and returned it to the client which simply displayed it.  
  
Some other ideas i had originally played around with for this project was opening multiple sockets for the adminclient, client, and server and having a separate socket for send and receive on each. This would have allowed for true asynchonicity. I was a little confused about the requirements though and wasnt sure if we could open sockets out side of the spec so i dropped this idea. The other thought was to ObjectStreams to allow for serialization, i dont remember why i dropped this though.

### General CodeMigration:

I thought this topic was kind of interesting mainly in the fact that at my current company we are working on an Azure Cloud Platform that is built on top of the [Service Fabric Cluster](https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-deploy-anywhere). This also happens to be the platform that HALO the video game is built on.  
  
At a 1000 mile high view the Service Fabric Cluster is a cluster of nodes that run microservices. It has the capability of migrating which service is run on which node based on a set of rules that by default are determined by the resource availability on any given node at any given time. It has 2 service types, one is Stateless one is Stateful. For this outline I'll just talk about stateless services because they are far simpler. A statless service is exactly what it sounds like, a service that doesn't maintain state( Think of it like an API Gateway or Event Processor). In the SF Cluster world you can specify how many instances of this service you need then it would balance those instances across the nodes potentially shutting down the service in one node and then starting it up in completely different node based. This is pretty simple for stateless services because you dont have to worry about anything in memory.   
  
A stateful service is a little bit of a different animal but they way they get around it is by maintaining the state of the service in multiple locations, called replicas, then when the service has to move it already has all the state that the original node had when it starts up( Again this is the most basic terms ).

### General CodeMigration - Reply

If you have access to Pluralsite this is a really good lecture on SF. It also explains where it falls in between container services and functions.  
<https://app.pluralsight.com/library/courses/azure-service-fabric-programming-models/table-of-contents>