# Assignment Module 3 Knock Knock Application

*By Michael Floerchinger*

08/27/2017

Java IV Programming Assignment #3 Knock Knock Client Server Application

Assignment Description.

Using the multi-threaded version of the Knock Knock example as the basis for this assignment:

Create a simple Swing GUI to start and stop the clients and server and display the jokes. All of the interaction does not have to occur in this one GUI. You may open other simple GUIs to replicate the behavior of the client cmd window. I expect you will need one application GUI that starts and stops the server. It should not be possible to start a client unless the server has been started. When the server is stopped, it is not necessary or even desirable to try to stop the clients. The application UI should be capable of starting multiple clients and displaying them simultaneously.

Handle the exceptions thrown by the stream and socket classes.

Read the jokes from a resource such as a file

Successive clients should get different initial knock-knock jokes. You do not have to guarantee the difference, just the randomness of the initial joke. Consider a load time solution that populates the clues and answers arrays randomly. Also consider a run-time

solution that has a random start location and operates on the lists as circular lists.

Add appropriate Javadoc comments to the source code.

Make any other changes you think should be made to improve the "OO-ness" of the design (at least one). The example code cries out for moving code out of the main methods and into appropriate class methods. Please explain what changes you made in the comments while submitting your assignment.

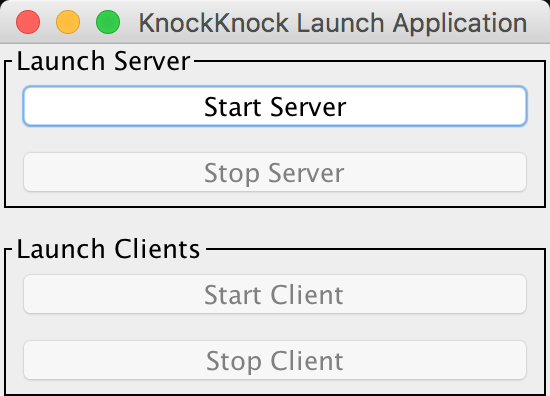
# Requirements and Deliverables:

***1. Create a simple Swing GUI to start/stop the clients and to display the jokes.***

The KnockKnock main application launches with a single central location for starting and stopping the application components consisting of:

* The server service listening on a TCP port
* The client application connecting to the server's TCP port

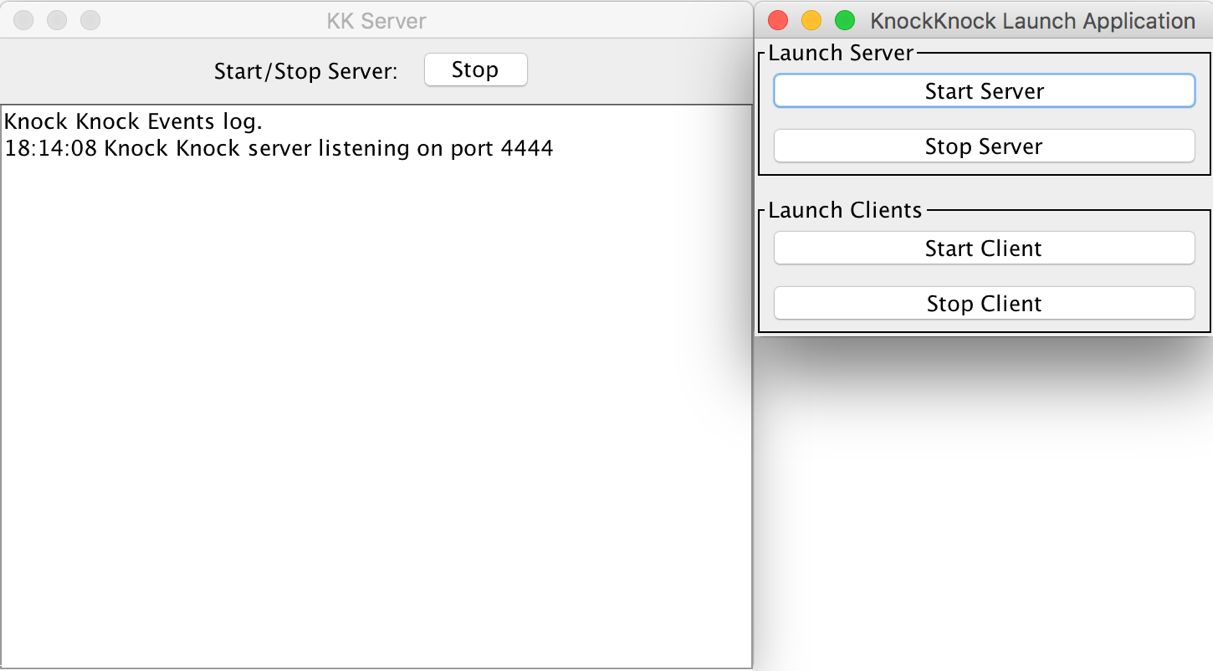
Main Knock Knock Application – start server and client applications



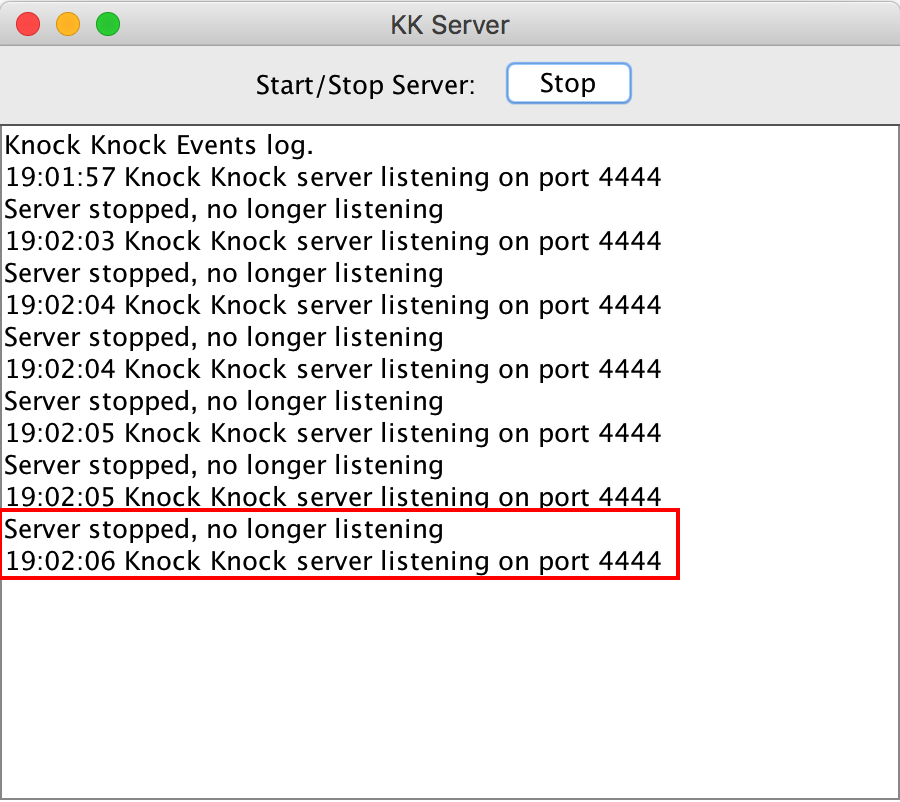
This is the screenshot of the application starting. This is the main application that enables you to start two additional functions:

* The server service
* The client service

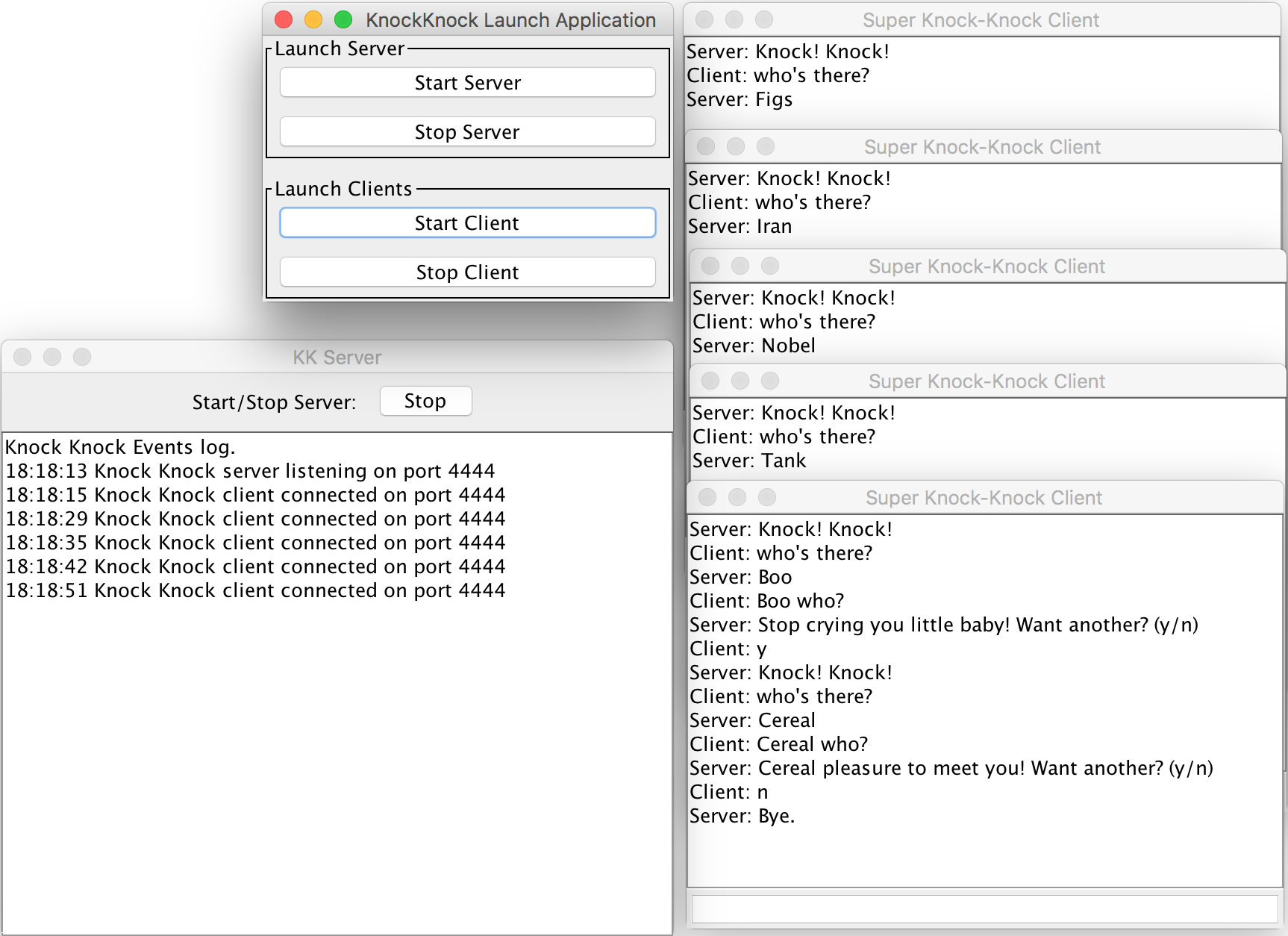
**Server service** – start and stop



Clicking on the “Start Server” button starts the server service on localhost listening on port 4444. The server service can be repeatedly started/stopped as many times as required. When the server is stopped no new client connections are possible.

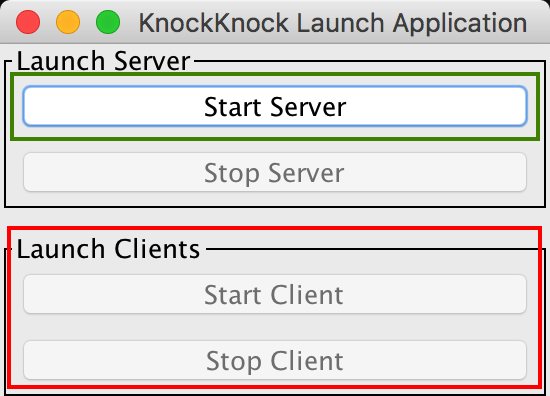


**Client Application** – start multiple clients

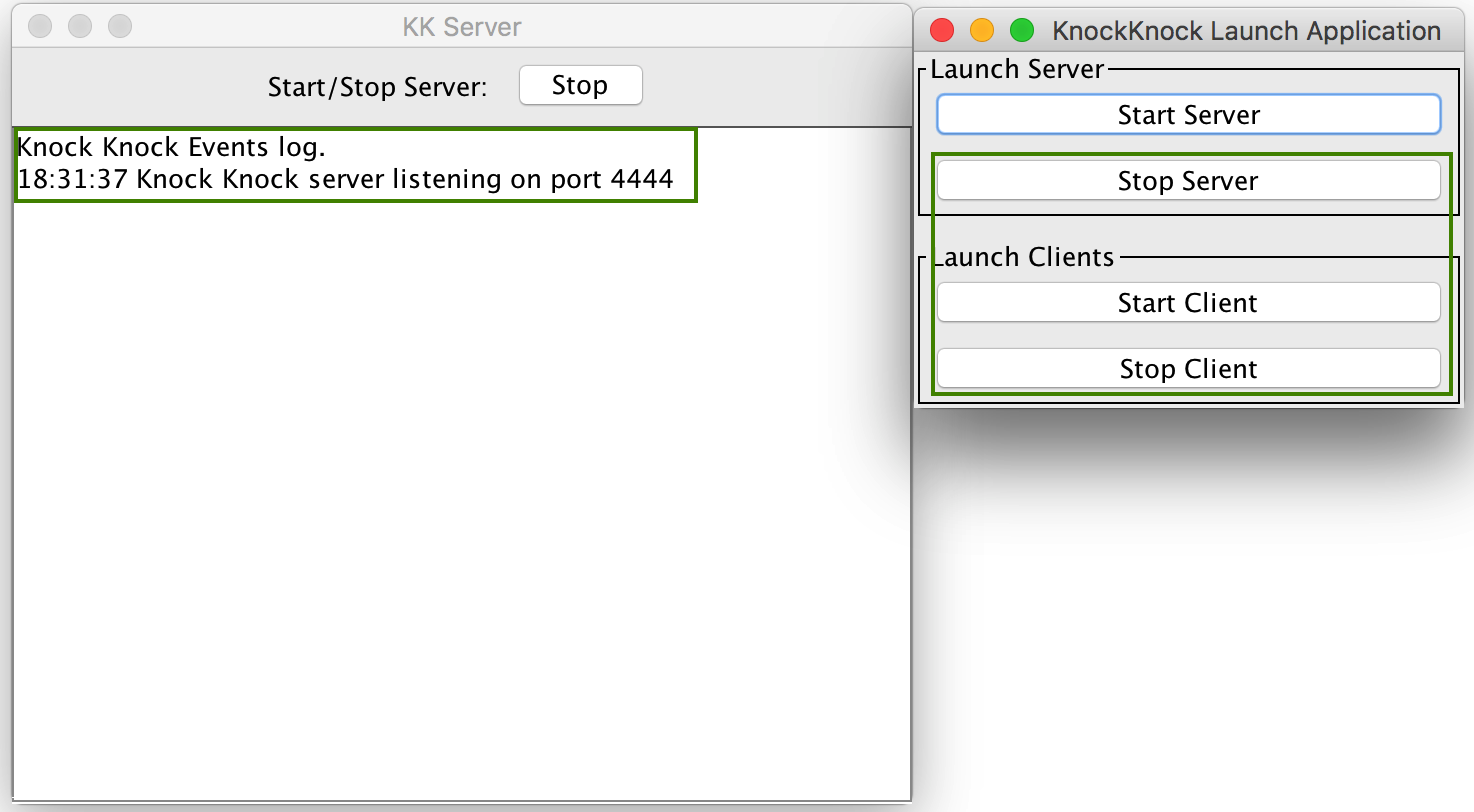


Starting the client records each clients connection to the server with a timestamp. Also the above screenshot shows many clients connecting and receiving different selection of questions.

***2. It should not be possible to start a client unless the server is running***



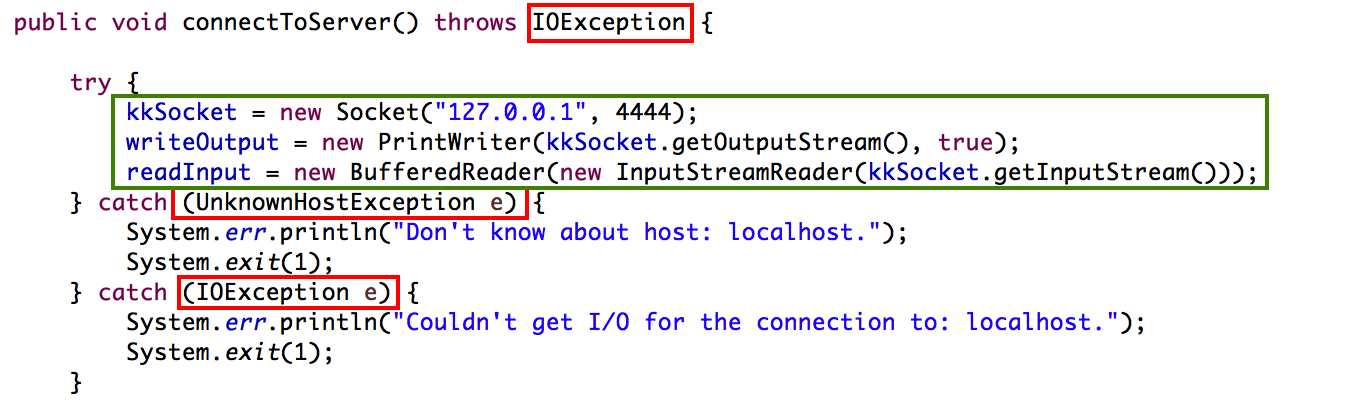
This screenshot shows that it is not possible to start the client until the server service has been started.



Once the server service is running and listening on TCP port 4444 the start/stop client buttons become active.

***3. "When the server is stopped, it is not necessary or even desirable to try to stop the clients." NOT DONE***

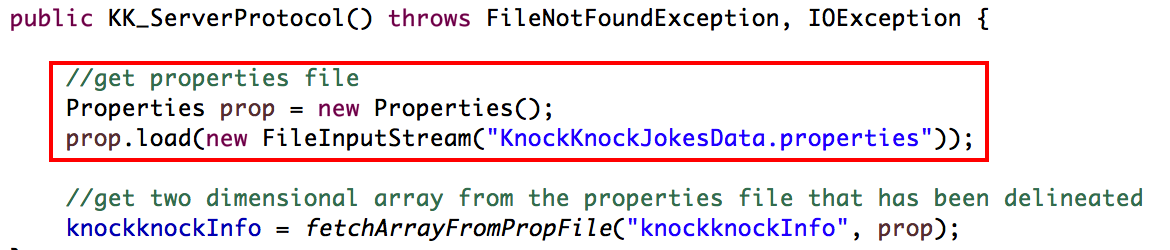
***4. Handling the exceptions thrown by the stream and socket classes***



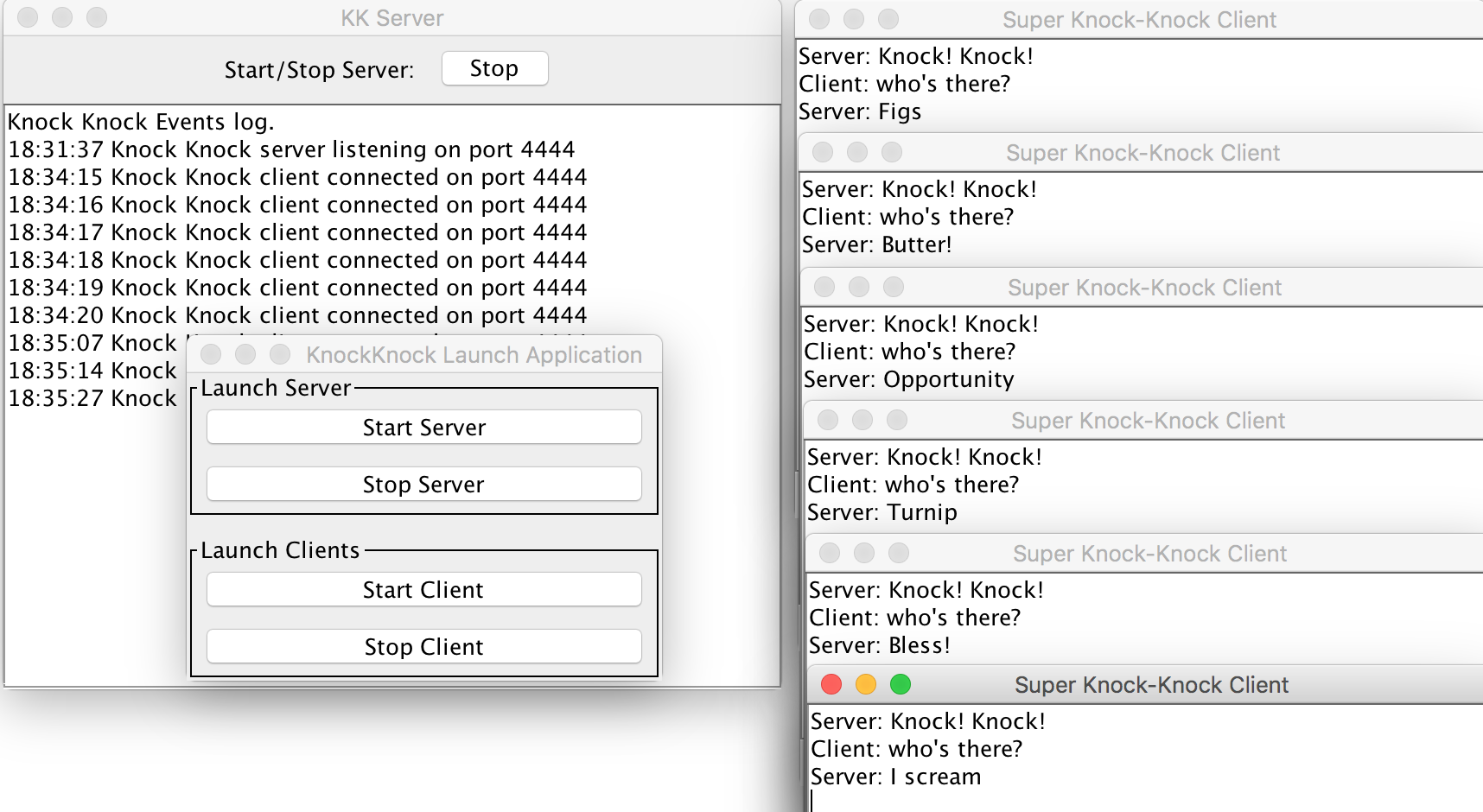
Depending on the situation, most of the exceptions thrown for streams are IOException errors. Additional ones include:

* FileNotFoundException
* UnknownHostException
* IOException

***5. Read jokes from are resource file***



***6. Successive clients get different jokes***

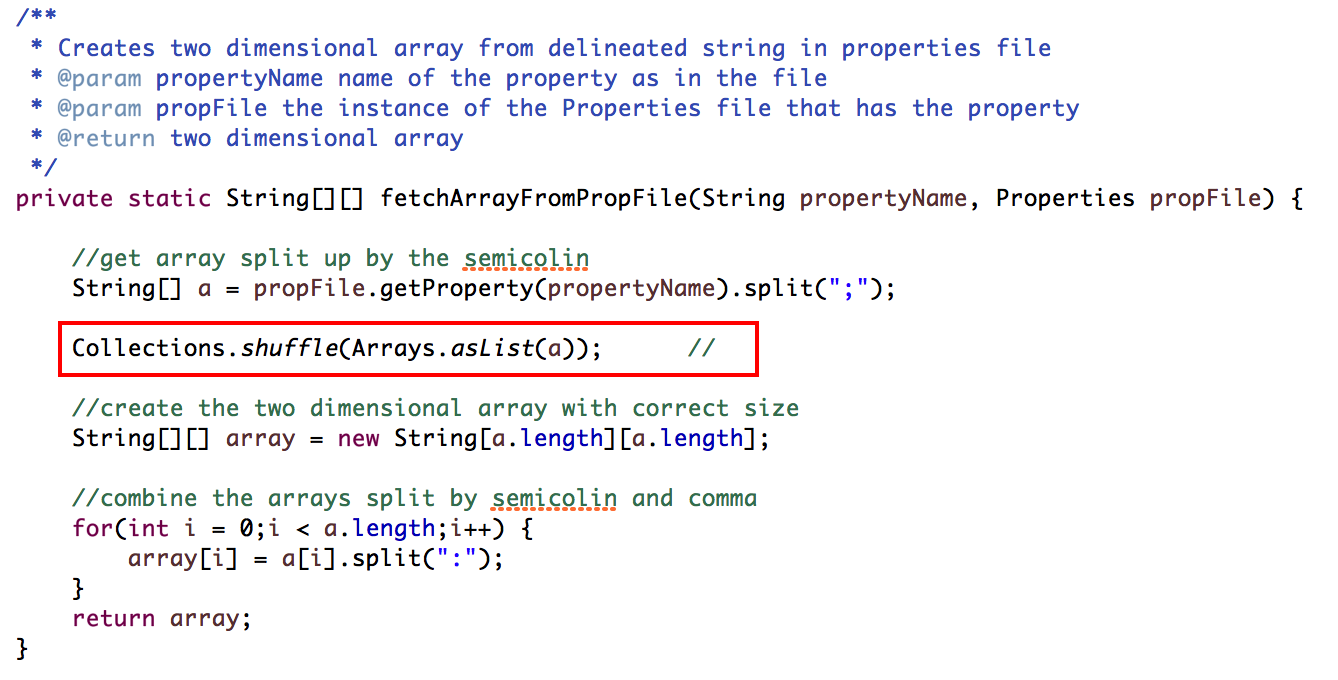


The above screenshot shows another random connection of clients receiving another random selection of knock knock jokes. This is different from the previously displayed screenshot where multiple clients connect to the same server. Additionally, and explanation of the randomness implementation is detailed in the next item.

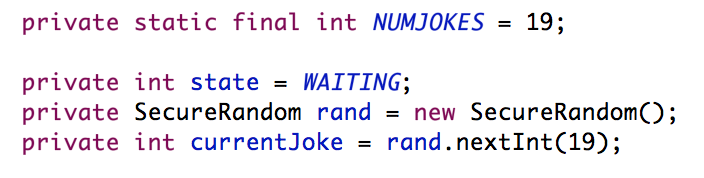
***7. Consider a load time solution that populates the clues and answers arrays randomly.***

The randomness implementation relies upon various strains of randomness as the clients connect. These are:

* A random number generator
* Randomizing (shuffle) location within the array

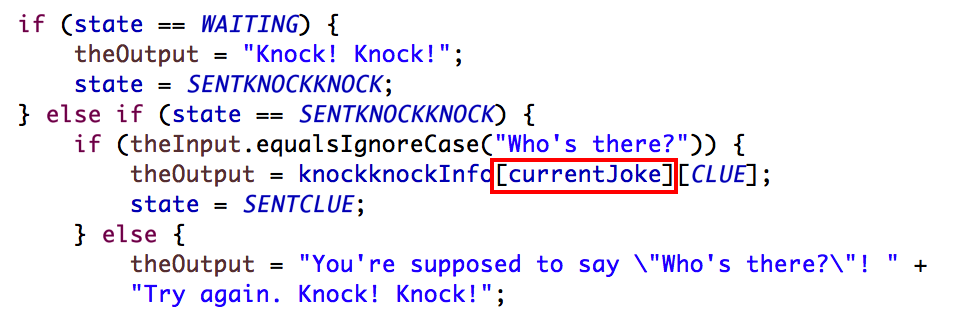


Publishes to an array and uses the Collections.shuffle() to randomize the collection inside the array.



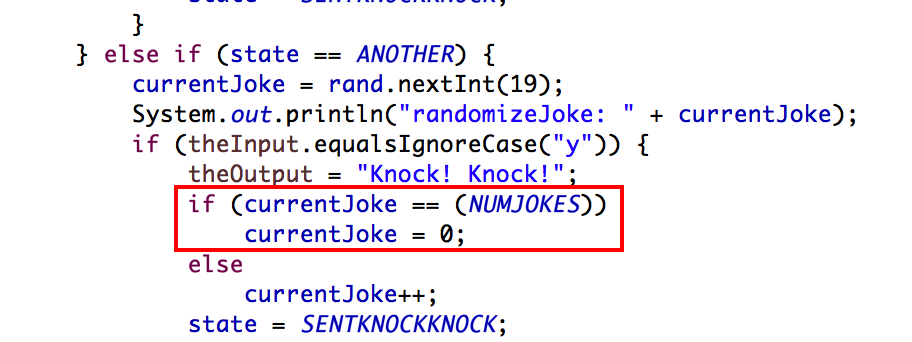
Every time a client connects it sets the currentJoke parameter with a random location from 0 – 18 inside the array.

The as the client completes the first response to “knock knock” the following code selects a randomly located location from the array.

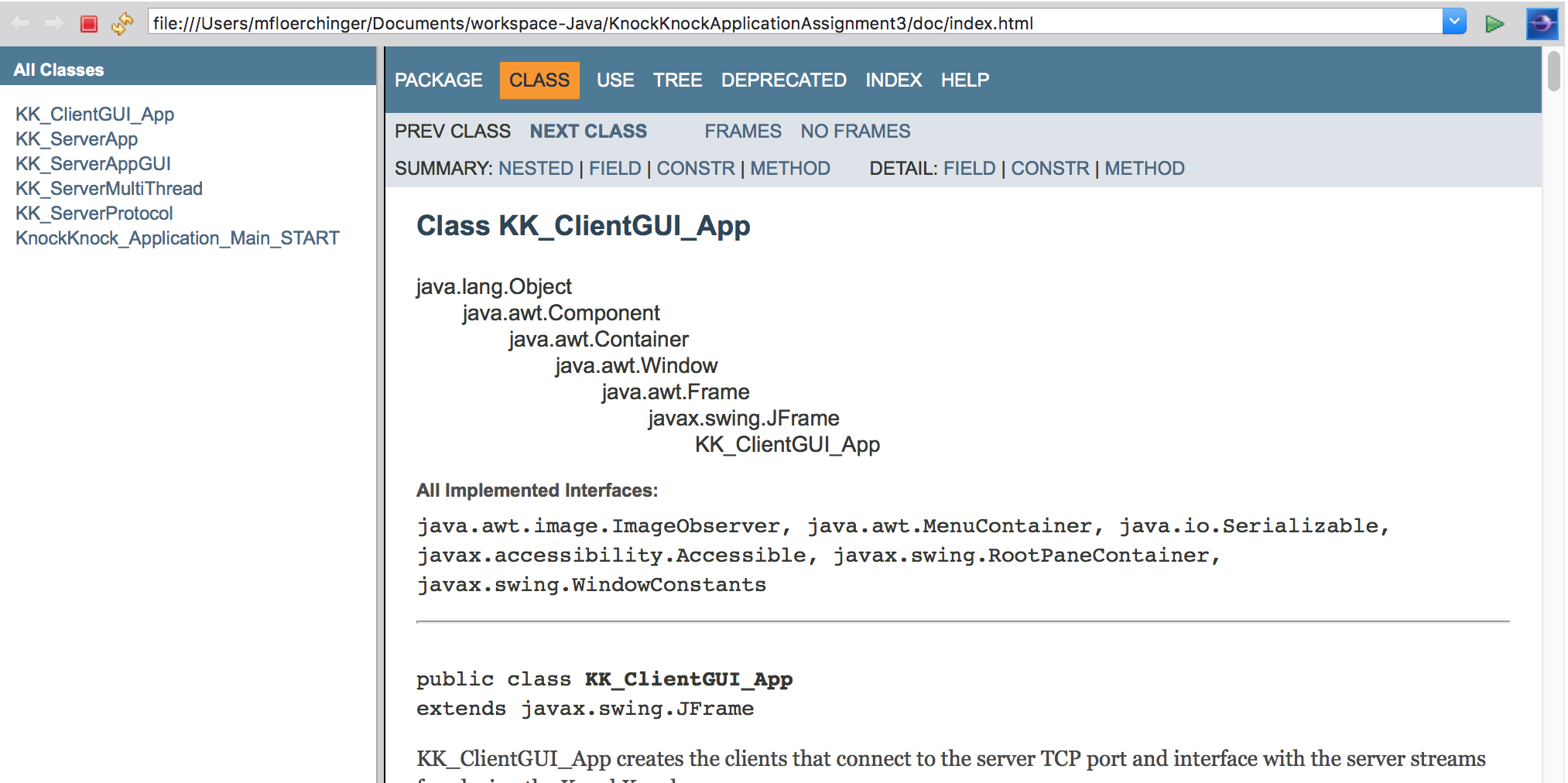


***8. Consider a run-time solutions that has a random start location and operates on the lists as circular lists => ring buffer***

Implementation of a “poor-mans” ring buffer by looping the index location once it reaches maximum to return to the beginning.



***9. Add Javadoc comments to the source code***



***10. Make other changes to improve the "OO-ness" of the app***

Implemented the application using the Model-View-Controller relationship, by breaking the components into view (swing UI), server logic (ServerApp), game logic (ServerProtocol).

