1902160

Implementation details

The Barista class is the server. Execute the Barista class and it will start a thread that runs the method `listenForCustomers()` which uses ServerSocket.accept() to create a new socket for Customers (clients) connecting via netcat or similar. Once a client connects, a new instance of customer is created with a new socket, and executed in a new thread.
The thread that is listening for clients then instantiates a new thread that runs the method `listenForRequests(Customer customer)` that scans for input from the client, interprets the input into requests, and calls methods in Barista as appropriate.
The methods available include:

`addOrder(Customer cust, int teas, int coffees)`

This method is called from `listenForRequests(Customer customer)` and it adds a specified number of teas and/or coffees to a new or existing order.

`getOrderStatus(int ID)`

This method is called from `listenForRequests(Customer customer)` and it sends the order status to the customer/client that requested it via the `talkToCustomer(Socket socket, String message)` method.

`CallOutStatus()`

This method is called from various parts of Barista and it's methods. It sends the number of customers, and the contents of all the trays, to all customers.

`talkToCustomer(Socket socket, String message)`

This method is called from various parts of Barista and it's methods. It sends the specified message to the specified socket. It is used to communicate with specific customers.

`formatOrder(int teas, int coffees)`

This method is used to format output to the customer in the form: x Tea(s) and x Coffee(s).

`trayContents(int ID)`

This method is used to format the contents of the 3 trays into a string to be output to the customer.

Usage:

After Executing the Barista class, and with netcat installed, you can type `nc localhost 8888` into one or more new terminal windows. You will then be prompted to give a name, and order drinks with the format “X Drink” where ‘X’ is an integer and ‘Drink’ is tea(s) or coffee(s). or “X Drink and X Drink”, or simply “X drink X drink”.

There is always a listenForCustomers thread running, and for each client that connects, there is a new Customer thread, and a new ListenForRequests thread. So the number of threads being run by Barista is n*2+1 where n is the number of connected clients.
Atomic variables for variables that were being accessed by threads to avoid blocking and concurrency errors.

bonus features:

Lots of input handling for client input, the strings and checked for irregular patterns of words separated by spaces. I think I may have forgotten to make it so that you can’t add negative integers (i.e. -2 coffees), but aside from that, the input checking is fairly robust.

Project review and personal reflection

The first draft of my program, had direct calls from Client to methods in Barista. It worked as intended, but was not purely client/server architecture. I had a lot of trouble making the program work using only sockets for communication between the server and client.
The final draft uses only sockets for communication between the client and server, and although it took a long time, I am proud of it.
I didn’t have any time left after making it work that way, to attempt the bonus tasks in the brief, which is unfortunate, as I would have liked to have attempted them. I worked right up until the deadline, having only had time to start a week prior. My pace was consistantly strong through the week, but it just wasn’t quite as much time as I would have liked.

After submission, I noted that a few of the methods, particularly getOrderStatus(int ID) and trayContents(int ID), could, instead of taking an int ID, and searching for the customer and order associated with that ID, it might be preferable to simply pass the Customer instance that called the method, as this may avoid some looping, and they are both called from listenForRequests(Customer customer), which has access to the Customer instance via it’s parameter.

I feel that the architecture is laid out in an unusual way because I started the project with a limited understanding of how sockets work, but came to understand them, as well as threads, much better over the course of the project, and would do it differently now.