CE303 assignment: The 'Virtual Café'.

1. Objective of the assignment

The objective of the assignment is to build a "Virtual Café" system, using a client-server architecture.

Each client application will play the role of a `Customer` who wants to order tea or coffee. The server application will play the role of the café's virtual `Barista`, who processes the orders, prepares tea and coffee, and delivers orders back to the customers.

2. Deliverables and breakdown of marks

You will need to submit two things:

- Code, implementing the VirtualCafé system, as per the specification below in Section 3: "Code Specification".
  - *This will count for 75% of your mark.*
- One report, no more than 1000 words, discussing your implementation choices. See Section 4: "Report specification".
  - *This will count for 25% of your mark.*

3. Code specification

Submitted files

You should submit a zipfile with the following contents:

- A file called `Barista.java` with a valid `public static void main` method, corresponding to your server application.
- A file called `Customer.java` with a valid `public static void main` method, corresponding to your client application.
- Optionally, if you write any more .java source code that you call from within `Barista` or `Customer`, then this should be placed in a folder (i.e. package) called `Helpers`
- If using the `gson` library, then your zipfile archive should also contain a `gson.jar` file at the root of the archive.
- It should be possible to compile your server application on a terminal by using:

  ```
  javac -cp "." Barista.java
  ```

  (or, e.g., `java -cp ".:gson.jar" Barista` if you've also included `gson.jar` in your zipfile),
  and then to be able to run it using

  ```
  java -cp "." Barista
  ```

  (or `java -cp ".:gson.jar" Barista` respectively).

  > *Note: on windows one should use a semicolon (i.e. `;`) instead of a colon (`:`) for the classpath.*

- Similarly, you should be able to compile and run your client application using equivalent `Customer` commands.

Note that the intent is to be able to run one `Barista` server and multiple `Customer` clients simultaneously, as separate processes (i.e. each from their own terminal).

Role of the `Customer` client

The client will be able to perform the following functions:

- The user will be able to type commands in a terminal, that the client will send to the server in order to order tea or coffee, ask for their order status, or leave the café.
- The client should also be able to print to the terminal any messages or replies sent to them by the Barista.
- The user can order one or more teas or coffees via the client at any time, by typing commands like the following:
  - `order 1 tea`
  - `order 1 coffee`
  - `order 1 tea and 1 coffee`
  - `order 2 teas and 3 coffees`
  - `order 1 coffee and 4 teas`
  - ... etc
- The user will be able to inquire about the status of their order at any time, by sending the command `order status` to the server via the client.
- The user will be able to leave the café at any time, by typing `exit`.

Bonus points awarded if:

- Your client can also intercept and gracefully handle SIGTERM signals (e.g. pressing Ctrl-C) for exiting the café.
- Your client can receive messages from the server at any time. Not just as a response to a command sent by the client.

Role of the `Barista` server

The server should perform the following functions:

- It oversees three different types of areas in the café (which should be implemented using an appropriate number of appropriate data structures of your choice):
  - The waiting area where orders have been placed, and are waiting to be processed
  - The brewing area where teas and coffees get prepared.
  - The tray area where finished teas and coffees are placed temporarily, until orders are complete.

  At any point in time, the server should know how many items are in each of these areas.
- The brewing area should work as follows:
  - The server should only have capacity to process 2 teas at any one time, and 2 coffees at any one time. Tea and coffee preparation are independent from each other (so you could be brewing a maximum of 2 teas and 2 coffees at any point in time).
  - Tea preparation takes 30 seconds, coffee takes 45 seconds. (You should simulate these with an appropriate 'sleep' function).
- The server should listen for, and accept, incoming connections from clients (i.e. `Customer` program instances, running on separate terminals). In other words, 'customers' should be able to join the 'café'.
- *When* a client connects, the server should ask for a name, (which the user will be expected to type). The server will be using this name for all subsequent transactions with this client, until the client leaves the café.
- The server will listen for incoming orders/commands from connected clients (which should follow the command format specified previously in the "Role of the Customer client" section).

  At any point, if the server receives an unknown command, it should respond to the client by sending back an appropriate error message.
- Once a server receives an order, it should acknowledge this to the client by name, by sending back a message like:
  - `order received for Mary (2 teas and 3 coffees)`
  - `order received for John (1 coffee)`
  - ... etc
- The server should know at any point in time how many clients are in the café, and for each client it should know if the client is waiting for an order, or if they're just idly sitting in the café.
- When clients "exit" the café, this should be handled appropriately (whether this was because of an `exit` command, or due to client disconnection).
- The server should respond to `order status` commands by clients, by sending back a message like:

  ```
  Order status for John:
  - 1 coffee and 2 teas in waiting area
  - 1 coffee and 1 tea currently being prepared
  - 2 coffees currently in the tray
  ```

  (with no message for an area if there is nothing happening there for that client).

  If the client issued an `order status` command while in an "idle" state, the server should *simply* reply, e.g. "No order found for John".
- Once a client's order has been completed (i.e. all items for that client are in the tray area), the server will inform the client by sending back a message, such as:

  ```
  order delivered to John (2 teas and 1 coffee)
  ```

  and then remove the relevant items from the tray area.
- If a client orders new items before their previous order has completed, instead of generating a new 'order', the new items should simply be added to the existing order.
- If a client leaves the café before their order has completed, simply discard the relevant objects from the respective areas.
  (But, see bonus points below).
- Every time the server changes state in any way, it should output a log of the current state in the terminal, consisting of the following items:
  - Number of clients in the café
  - Number of clients waiting for orders.
  - Number and type of items in waiting area, brewing area, and tray area.

  (Also see bonus points below)

Bonus points awarded if:

- Whenever a client leaves, the server should check if any of their brewing or tray items can be repurposed for orders belonging to other clients.

  If this happens, print appropriate messages in the server's terminal, e.g.
  - `1 coffee currently brewing for John has been transferred to Michael's order`
  - `1 tea in John's tray has been transferred to Michael's tray`
  - ... etc
- In addition to logging status updates in the server's terminal, also keep logs in an appropriate JSON file. Each entry should also be associated with the date and time of the event.

Other remarks:

- You are expected to use appropriate data structures to handle the waiting, brewing, and tray "areas".
- It is expected that the server and clients will communicate through sockets
- It is expected that the server will be able to handle multiple clients *concurrently* (i.e. using threads).
- It is expected that orders will be synchronised appropriately to avoid race conditions
- All code submitted should be well documented, readable, maintainable code.
- In theory, it is possible to achieve full marks if all parts above have been implemented successfully as per the specification (and code is of excellent quality).
  Bonus points are awarded *on top* of this mark, and may earn you up to an extra 10% added to your final mark
  (so, in other words, in theory it is possible for 'code + report' to achieve a maximum score of 110%).

4. Report specification

The report should be no more than 1000 words, and should consist of the following sections:

Implementation details

This should detail how you implemented your server and client architecture. It should be as brief as possible, but contain enough detail so that another developer reading this would have enough information to be able to implement your architecture. Specifically, make sure to mention which data structures you used, and why you chose them; and what threads are running in your server and client classes, what their role is, and how they achieve this.

If you have implemented any of the bonus features, mention this here as well.

For any functionality mentioned in the code specification that was not implemented, or only partially implemented, mention these in a separate section, explaining why this was the case. Insightful explanations indicating legitimate technical challenges faced and lessons learned may be awarded partial marks. Failing to report missing or partially implemented functionality in this section will be penalised.

Project review and personal reflection

Reflect on your work. E.g. How did it go? Which parts were challenging? Are there any features that you are particularly proud of (and why?). Could you improve on your current design / is there anything you would do differently next time or if you had fewer time constraints? How did you do with planning? Did you have a steady pace, or did you over/underestimate time required for particular tasks? Did you stick to your original plan, or did you have to change things half way? Have you learnt any lessons while doing this assignment that you think will help you as a programmer going forward?

That is all. Good luck! :)