**Report Specification**

**Completed Requirements/Implementation details:**

* Program compiles and runs with javac -cp “.:gson.jar” \*.java -> [NOTE: it shows warnings as I am using an external library BUT the code should still run]
* All supporting classes are found in the Helpers package/folder.
* The Customer class represents the Client and functions through the of CustomerProcessor class [+ listenerThread]
* The customer is initially asked for their name upon entrance, which is sent to the server and saved on the client program simultaneously.
* The customer client then connects to the server (Barista) class and establishes successful connection upon prompt from the server.
* The customer is then able to order drinks, check the status of their order(s) or leave the café and can successfully receive corresponding terminal responses.
  + E.g. When the customer orders, their name and order is confirmed to them from the server.
* The customer can order at any time if they have not left the café.
* [BONUS] SIGTERM signals are successfully and gracefully handled using an external Signal class
* [BONUS] Always receiving messages from server almost working -> CHECK NOTES
* The Barista class first creates a server instance (allowing incoming connection(s) with multiple clients. A server-universal café and appropriate threads are started to complete initialisation.
* The café uses Integer ArrayLists for the different working areas in the café [for simplicity as each drink simply has an ID] to streamline drink processing. Maps are present for objects in the café to retain information (i.e. customers, orders or drinks)
* At any changing state of the café, a log is produced and printed. This information includes the number and type/status of objects throughout the café-> NOTE: it is currently commented out to save terminal writespace and can be seen in JSON
  + [BONUS] This information is also logged into myJSON.json every 30 seconds -> CHECK NOTES
* The customerName is received and saved from the client on placement of an order for future reference.
* An orderNumber is generated, on receipt of an order from the client, before drink threads are generated and processed by the café. Appropriate threads are started using regex Matcher and Pattern classes to decode the client order. The server confirms the order once processed
* orderThread is responsible for the creation of an order from server
  + teaThread and coffeeThread are responsible for drink brewing.
* The server can successfully effectively handle ‘order status’ commands by returning any drinks that the customer has ordered (unless they have no orders where it will return null)
* If the client orders while they have an order waiting, their new order will be added to the current order instead of a new one being generated.
* If the ‘exit’ command is received from the client, the server will remove any related drinks and customer information from the café.
  + [BONUS] The café will also attempt to reassign any removing drinks to waiting drinks but -> CHECK REVIEW
* If the server receives an unrecognised command, an appropriate error message is sent back to the client
* When the server completes the order, it will return a message in the terminal to say that the order is complete -> CHECK NOTES
* Methods that work with different working areas within the café are generally synchronised to prevent multi-threading bugs arising by only allowing a single thread to access (and change) any one area at a time.
* The café will only brew two of any kind of drink at any moment of time. It takes 30 and 45 seconds to brew tea and coffee, respectively, and this is done through the use of Thread.sleep()

**Notes/Debugging:**

* [BONUS] Customer client always receiving messages -> I tried a variation of multiple solutions – although I could not produce one without affecting the flow of the program. The problem primarily comes from the fact that the client must either listen to the server or the client (customer) [reader vs System.in]. The closest solution involved creating and starting a separate thread (you can see progress in listenerThread.java and CustomerProcessor.listen()), but for some reason, reader.hasNextLine() would keep waiting for the server instead of breaking out of loop. Hence you can see ‘pause’/‘resume’ method in CustomerProcessor to imitate the deprecated Thread.suspend/resume.
* [BONUS]JSON -> the logging occurs at every 30 seconds as I never got round to fully implementing GSON/JSON inside the Café (as it ideally should stay) [I believe I have a bootleg method but would appreciate feedback], otherwise, I would try implement serialising/deserialising Café, converting with gson to json, then writing at every state change.
* ‘order status’ + ‘idle’ -> Currently, my ‘order status’ command from client will return an appropriate message if there are no current but, I could have implement this requirement fully by checking if customers were ‘idle’ although I was not 100% sure that my customer states were being changed correctly (idle/waiting) so I kept the old model.
* Complete order status -> The customer does not currently receive order completion confirmation -> BUT once the client always able to receive messages is fixed, this bug will also be fixed.

**Project Review**

* [BONUS] SIGTERM -> I understood the main implementation of this was the ‘ShutdownHook’ although I could not get it to function as requested because I did not understand how to link the SdHook to the SIGTERM signal -> ultimately the Signal class did the job effortlessly.
* ‘exit’ command -> I have implemented a drink ‘removal’+‘reassigning’ method into the café, although I seem to be having threading problems, where because most methods are synchronised, the removal method cannot access areas as they are being used by the threads (except for waiting).

General:

A very engaging and difficult task which seemed simple but held many challenges. My knowledge on the JVM and Java functionality has vastly grown, more specifically information on Concurrency as a whole. I have restarted over three times and steered far from the original plan.