University of Essex

Year 3

The ‘Virtual Café’

Student Number: 1905341

CE303-6 – Advanced Programming

Tutor: Dr. Tasos Papastylianou

Date: 02/12/2022

Contents

List of Figures

1. Implementation Details

As far as I can see the project fully meets the code specification (including bonuses), see the final file structure below. Code complies successfully on windows lab PC, unable to access Linux to test.

Figure 1: Final file structure.

Data Structures

Many different existing and custom data structures were used.

Existing data structures:

- Array lists were used for the waiting/brewing/tray areas. A queue was initially used, however, the ability to remove a specific element from the collection became more useful than the benefits that queues offer.

- Maps were very useful when manipulating data using streams, for example, an order status response needed the number and type of each drink inside each area. The below code populates a map that contains a Long (count of drinks), DrinkType (custom Enum of drink types) and, a String (area). The message for order status can be generated using this data structure easily, another benefit of this design is that this code will work even if a new drink gets added to the café.

```
Map<String, Map<DrinkType, Long>> numberOfDrinksByDrinkTypeByArea;

//Count number of drinks in each area
//Mapped by status (area) and type of drink
numberOfDrinksByDrinkTypeByArea = client.getDrinks().stream()
        .collect(Collectors.groupingBy(Drink::getStatus,
                Collectors.groupingBy(Drink::getType, Collectors.counting())));
```

Figure 2: Map and stream usage.

Custom data structures/types:

- Message – Holds information relevant to a message sent between the server and clients.
- MessageCommand – Enum that holds all possible commands for a Message.
- Log – Holds the café status information.
- Drink – A parent class that is inherited by Tea and Coffee and holds information regarding a drink.
- DrinkType – A custom enum with methods to help with types of drinks.

Threads

Client

Each client contains three threads: the main thread is used up for performing the connection handshake, initializing the client helpers, and then constantly reading inputs from the user, another thread is used by the ServerListener for listening for messages being sent from the server so it can receive messages at any time, looping while the connection is still valid, and the last thread is delegated to the shutdown hook inside of ServerInterface, which runs a block of code to let the server and the user know that the client is exiting, this happens when either the client closes normally, or a SIGTERM signal is received.

Server

When starting the server will contain only two threads, the main thread is listening constantly for new connections, when a new client connects the server will create a new ClientInterface which implements Runnable so this is created on a new thread that will perform the handshake, upon a successful handshake the thread will listen for incoming messages from the client while the connection is valid, if the client leaves or disconnects the ClientInterface thread will die.

A second server thread is created when the server creates the café. Once the café has been initialized the server starts the startBrewing thread that continuously monitors the waiting area for incoming drinks, if any appear it will check if they can be brewed. If the new drink can be brewed (the server is not already brewing two of that drink), then the startBrewing thread will create a brewDrink thread for this drink which uses Thread.sleep() to simulate the brewing, after this, the drink gets moved to the tray area, checkOrderComplete() gets called and finally the thread will die. This means that there can be anywhere from zero to four extra threads on the server to handle the brewing.

Total server threads at any time: 2 + number of clients + (0-4)

Since multiple threads are using a shared data structure a ReentrantLock was used to lock/unlock at specific points to avoid race conditions in Cafe.

Bonuses

When the server gets informed that a client has left, it will remove all their drinks in the waiting area and it will try to redistribute the leaving client's drinks in the tray and brewing areas to other clients, by removing the client's drinks in the waiting area and reassigning the leaving client's drink to the client who is staying.

Every time the state changes in any way, the server will print out a log and write it to the Log.json file.

```
Adding 1 coffee to the brewing area for 3:Dave - Start brewing
Current cafe status: 02/12/2022 10:30:03.080
Number of clients: 1
Number of clients waiting for orders: 1
Waiting area:  1 coffee
Brewing area:  1 tea and 2 coffees
Tray area:     no drinks
```

Figure 4: Log in server terminal.

Figure 3: Json representation of the log.

2. Project review and personal reflection

I enjoyed this project and liked being able to design and code it, I feel like I almost got it to a completed project, the only change I think I would make, if time allowed, is to change the input validation to allow multiple chained orders in one command e.g

order 1 tea and 3 coffees and 6 waters… etc

I did struggle with getting the redistributing drinks for leaving clients working perfectly as I was getting the drink of the client who was leaving mixed up with the other client.

Features I am proud of

The code is well documented and has a JavaDocs documentation alongside it (this can be found in the index.html file in “VirtualCafe Documentation”).

The way that this project is designed and coded means that it is very scalable, in terms of adding new drinks as only a few changes to the code is required to integrate a new drink, they can be seen on the next page for a new drink: water

```
private void initDrinksSetup() {
    //Map used for checking a drink is valid by using isValidDrink(String)
    drinkTypeByString.put("TEA", TEA);
    drinkTypeByString.put("COFFEE", COFFEE);
    drinkTypeByString.put("WATER", WATER);//########################

    //Initialize current brewing map to 0
    for (DrinkType key: DrinkType.values()) {
        currentBrewingByDrinkType.put(key, 0);
    }

    //Set how many drinks can be made by the barista at a time
    maxBrewingCapacityByDrinkType.put(TEA, 2);
    maxBrewingCapacityByDrinkType.put(COFFEE, 2);
    maxBrewingCapacityByDrinkType.put(WATER, 2);//########################
```

Figure 6: Drinks setup in Cafe.

```
Extended data structure for Water.
public class Water extends Drink {

    Creates a Water instance setting the DrinkType and the brewingTime.
    public Water() {
        this.setType(DrinkType.WATER);
        this.setBrewingTimeInSeconds(10);
    }
}
```

Figure 5: New Water class.

```java
public enum DrinkType {
    TEA( typeOfDrink: "tea"),
    COFFEE( typeOfDrink: "coffee"),
    WATER( typeOfDrink: "water");//########################

    private final String typeOfDrink;

    Creates a drink type and updates the string representation of the type of drink.
    Params: typeOfDrink – the string of the type of drink
    DrinkType(String typeOfDrink) { this.typeOfDrink = typeOfDrink; }

    Returns a new instance of a Drink based on the drink type
    Returns: a new instance of Drink based on the drink type
    public Drink getDrink() {
        switch (this) {
            case TEA -> {
                return new Tea();
            }
            case COFFEE -> {
                return new Coffee();
            }
            case WATER -> {//########################
                return new Water();//########################
            }//########################
        }
        return null;
    }
```

Figure 7: Drinks setup in DrinkType.

My planning overall was good, perhaps did not start the project quite early enough and development was at a steady pace, however, I did keep deciding to change functionality and re-design/code methods creating backups as they changed.

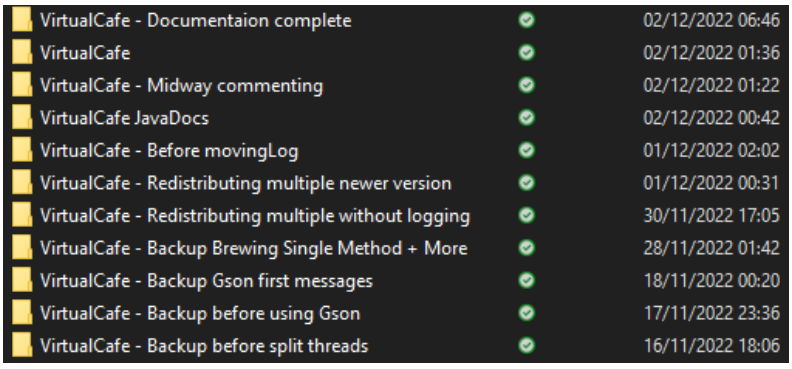

Figure 8: Backed up files for when a change was made.

I feel like I cannot describe all the features my project has within only 1,000 words and I have also tried to be as brief as possible in the report.

During this assignment, I have increased my portfolio of programming skills when manipulating data, specifically when using maps.