Advanced Software Engineering (F21AS): Group 4

Stage 2: Group Report

Dominic Taylor (H00246129)

Aina Centelles Tarrés (H00358958)

Leah Wright (H00360682)

Bo Han Jhan (H00317537)

Jiancheng Zhang (H00341619)

Table of Contents

1. Link to Repository

Clone with SSH: **git@gitlab-student.macs.hw.ac.uk:ase_coursework_group-4/coursework.git**

2. System Brief

This program meets the specification fully. All core requirements are met. Two extended functional requirements have been implemented:

1- The user can **alter the time it takes to process each item** of an order by using a JSlider in the GUI. So, depending on the number of items and the chosen time in the GUI, the processing time of an order will be:

orderProcessTime = nbOfItems * milisecondsPerItem [in miliseconds]

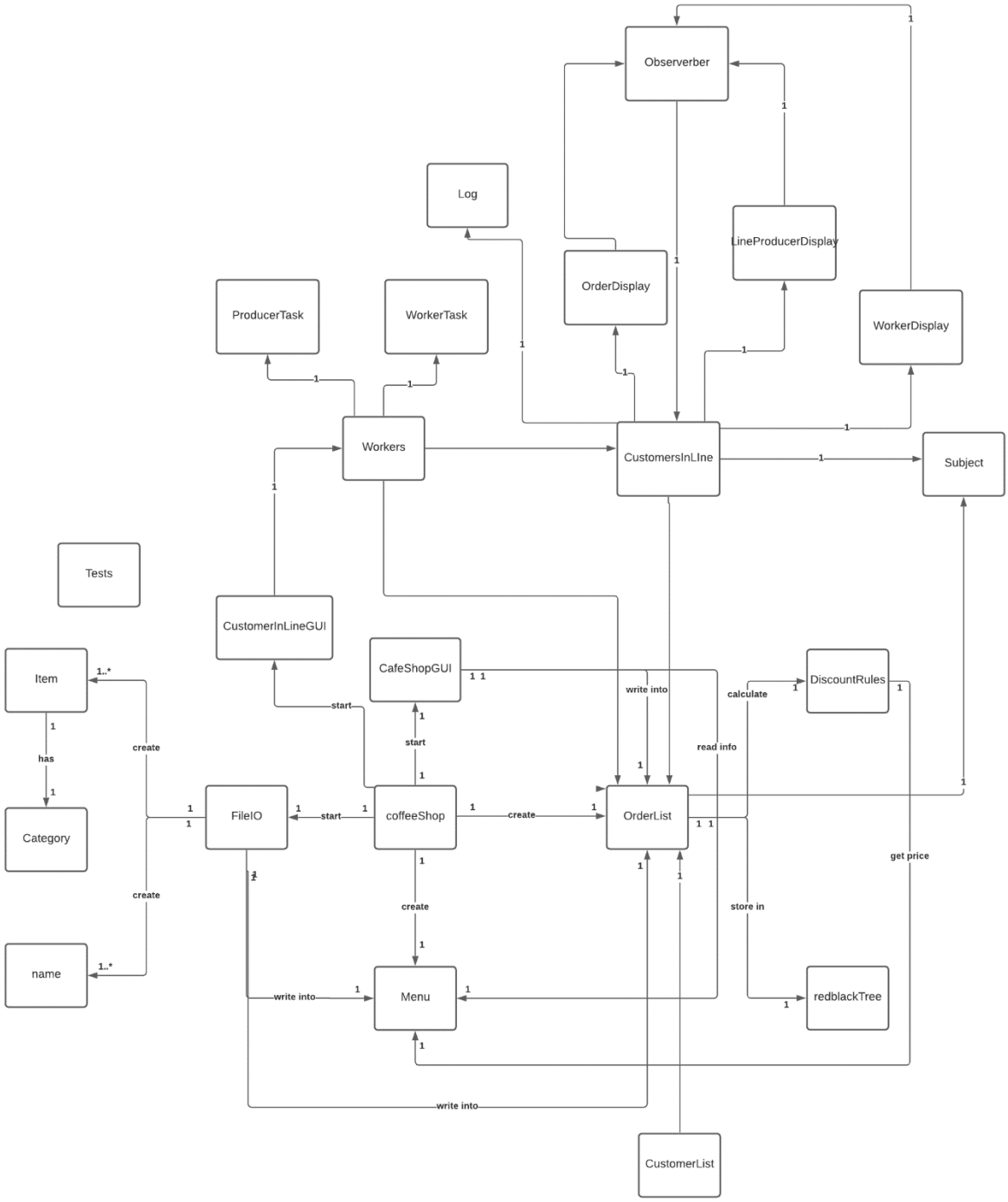
2 - **Staff members are added or removed depending on the length of the queue**. As a rule, for each serving staff there can be at most 2 people in the line. Thus, the number of required members of staff is calculated as:

requiredStaff = Math.ceil (queue_length/2)

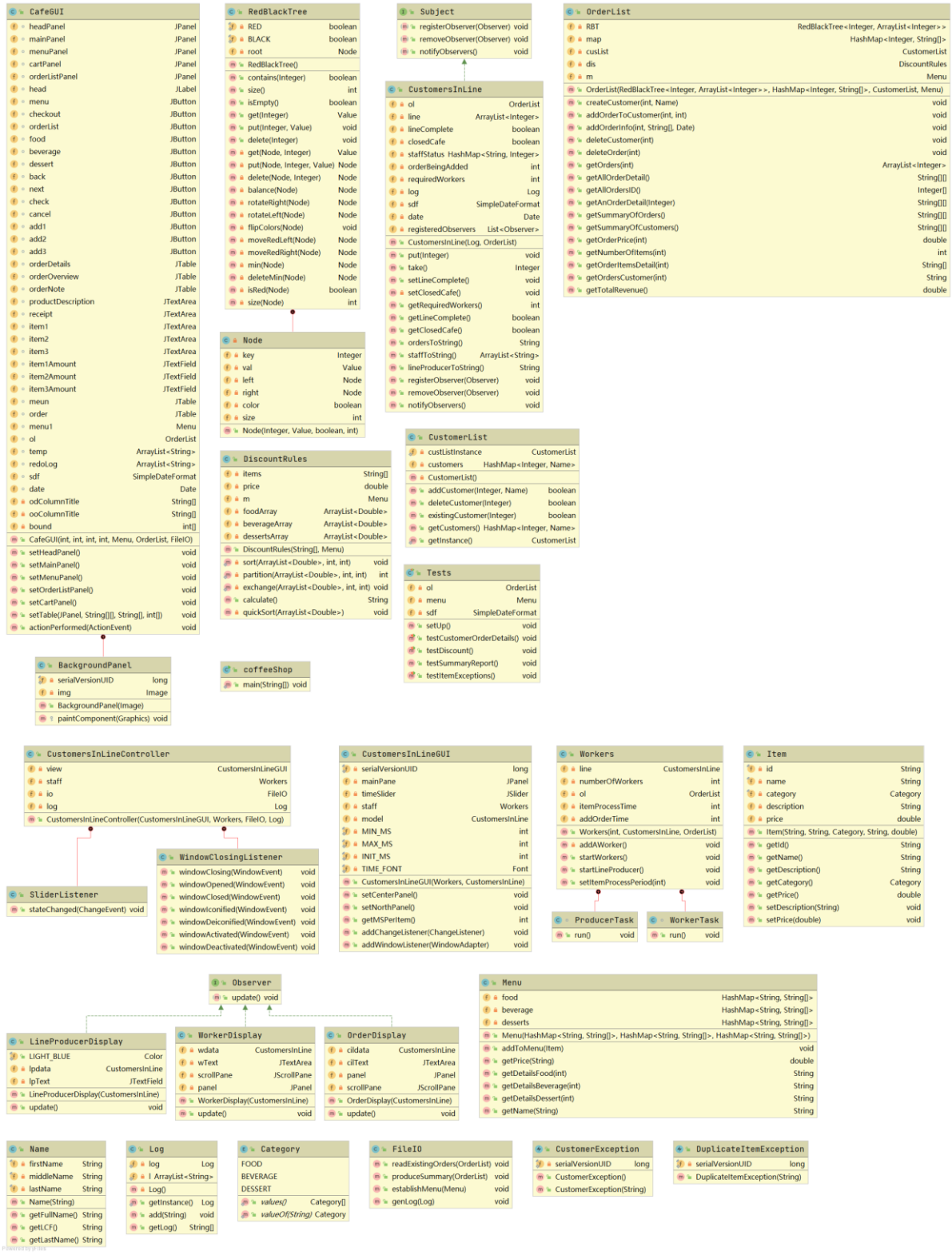
where Math.ceil returns the closest Integer that is equal or bigger to queue_length/2. If the number of requiredStaff is bigger that the active number of worker threads, a new thread is added. Similarly, if requiredStaff < activeThreads, threads go “on a break” and pause from processing orders. Even if the queue_length is 0, at least one worker (Worker-1) stays active at all times to close the cafe.

3. UML Diagrams

Class diagram



Class contents



4. Threads Implementation

The threads behaviour is managed in two classes: Workers class and CustomersInLine class.

Workers class

The workers class is in charge of two types of threads: **Worker threads** and a single **lineProducer thread**.

While the cafe is not closed, each **worker** thread fetches an order from the top of the queue using the method CustomersInLine.take() and then sleeps some time depending on the number of items in the order.

On the other hand, the **lineProducer** thread gradually adds the orders from the orderList to the queue using the method CustomersInLine.put(orderID). After adding each order, it checks whether more Workers are required and if so, calls a method to add a worker. Once all orders have been added, it signals to CustomersInLine that the line is now complete.

CustomersInLine class

In the **CustomersInLine class**, which deals with the queue of customers, two booleans are used for signaling of events: **lineComplete** (once all orders are added) and **closedCafe** (once line is complete and all orders have been processed).

take()

In the synchronized method **take()**, worker threads check whether they are currently required (depending on queue length). These threads are distinguished by their name, which looks like: Worker-x where x is a number, starting from 1. Threads are chosen to work or *go on a break* using this x number. If they aren't required to work, they are set to go on the waiting state. Once they are signalled to wake up, if the cafe is already closed they return.

On the other hand, if the line is empty, two things can happen:

· if the line is already complete —> this means all orders have been processed and thus the cafe is set to closed (in **setClosedCafe()**, all waiting threads are notified to be able to terminate) and the thread returns.

· if the line isn't complete —> more orders are coming so this thread is told to wait() until some order is there.

If there are orders to process, they take the one at the front of the line. The number of required workers is updated because the queue length has changed and all threads are notified because the number of required workers may have changed. Finally, the thread returns the orderID to process.

put()

The synchronized method **put()** is used by the lineProducer thread to add orders to the queue. After an order is added, the number of required worker threads is updated because the queue

length has changed and all waiting threads are notified because they may be required to work now.

other methods

This class is also the subject of the observer pattern used for the GUI. It has the methods **ordersToString()**, **staffToString()** and **lineProducerToString()** which inform the GUI of the state of the queue, each worker and line producer at any given time.

5. Design Patterns

One of the design patterns we have included in our application is the **observer pattern**. This pattern is used to allow many objects (observers) to register their interest in another objects state (subject). Within our application we have one subject: **CustomersInLine**, this contains the methods for adding and taking orders from the line. This class implements the **Subject** class, which allows it to register, remove and notify observers of when its state changes. We have 3 observers of the CustomersInLine Subject: **LineProducerDisplay**, **WorkerDisplay** and **OrderDisplay**. Each of the observers have an update method which will update based on CustomersInLine updates, they each implement the **Observer** class which provides this method. Each of the observers provides a panel for the GUI, therefore the GUI will constantly update as the CustomerInLine subject changes state.

The **Singleton pattern** is used in the log and CustomerList classes and the **MVC pattern** is also used in the architecture of the GUI code.

6. Program Management

Because the core concept of agile development is to focus on the interaction and communication between group members, it emphasises that an adequate management strategy has to be adapted to different personality of group member, which means that appropriate implementation/development plans will be formulated for different members according to what the time and place they can cooperate. Our group adheres to the belief of agile and believes that there is no approach can be identified as the best or unified standard but can be differentiated by appropriate and inappropriate.

During stage 2, our group had planned to have 4-6 cycles, depending on the limit time, for experimenting with agile methods. All the group members were assigned to act three different roles: product owner, scrum master and team. In order to skill each member of this program, they were in charge of different position in the iteratives. Again, to perform agile program more realistically we held scrum meetings twice a week incorporating with team member's personal time schedules. Especially, in the pandemic, our members were working together across different countries and time zones.

The activities performed in each iteration were 3-day scrum meeting, product backlog refinement, sprint planning, sprint, sprint review and sprint retrospective. Besides, all activities should be finished with time limitation. Take scrum meeting for example, every participant can only take 2-3 min for updating the group his/her job. The group wouldn't take more than 15 min in total on the meeting. Most of time, our group can have the meetings be controlled and end it in specific time limitation. However, sometimes development problems would prevent

the group from keeping focus on the purpose of the meeting. To fix this problem, group members were advised to manage their working notes before the meeting and to try to brief the working result as simple as possible. Since much work had been detailed in the working note, people don't need to pay too much effort on explaining their working progress but present key points of their work. In order to simplify the development process as possible as we can, we don't force everyone to post their working notes on our group chat instead we highly recommend our team members to take care of time in the meeting.

On the other hand, the biggest problem we had encountered is that the product owner might not be able to analyse the market and customer demand as professionally as a program manager since our team is not a real business organisation at all as well as because we rotated to play different roles in the program. The product owner would tend to stand in an engineer's stance and consider the level of difficulty first as prioritising items in the product backlog. Nevertheless, we eventually found out how to refine our agile process as stepping into next iterative. We figured out that nothing is more important than meet user's requirements and the team must stick the development with agile principles.

Finally, although our team implemented a few extend features in the end of this stage, at least, the basic functional requirements are all met under our teamwork. It is hard to identifying whether we have performed our agile development successfully because a few achievements had been accomplished in the program. However, if the success will be determined by whether the group adhere to the principles of agile in the program, the positive answer can be confirmed. The purpose of agile is to work close with customers and emphases the quality of our product rather than to complete a work quickly without thinking tightly about the user requirements.

7. Conclusion

Compared with the traditional waterfall development process, which we were adopting in stage 1, the strategy encourages that analyses all, designs all, implement all, and tests all, the experiment of agile in this stage introduces an iterative development process adopting partial analysis, partial design, partial implementation, and partial test in cycles. The advantage of latter is that we can implement the entire system architecture during the first few cycles of development to verify the feasibility of the architecture design, so the technical risks can be transferred to the early stage of development and the architecture problems can be found early.

Appendix I - Screen Shots

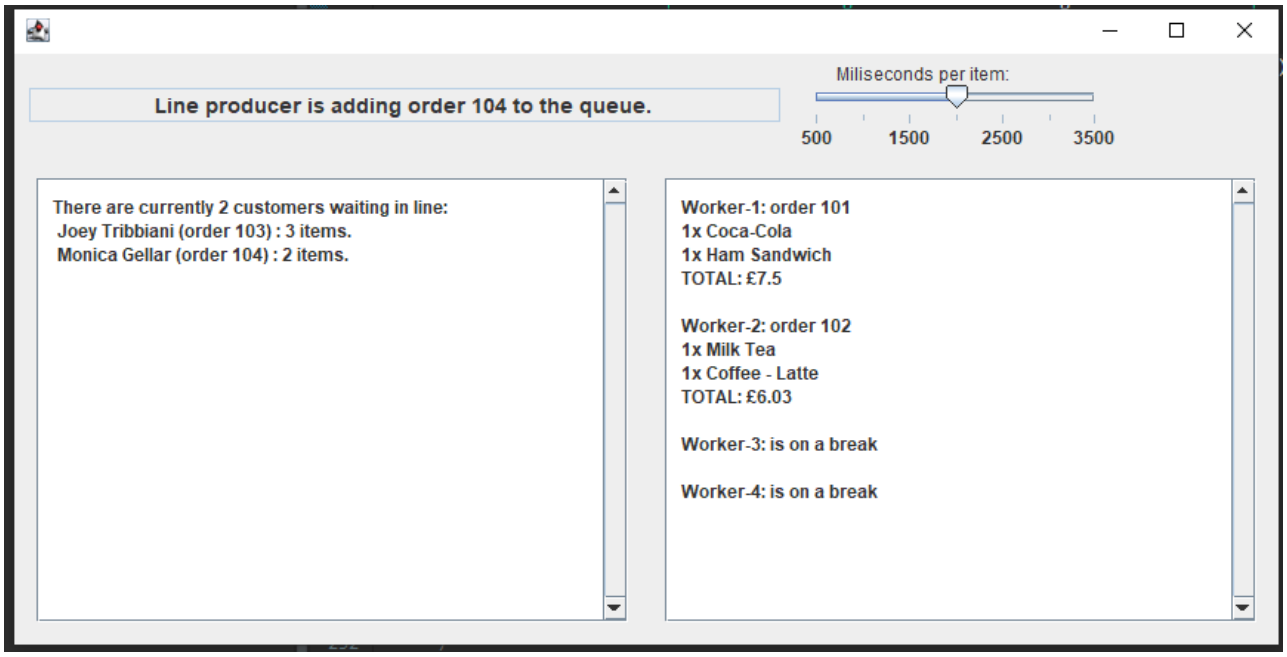


Figure 1: Start-up of GUI

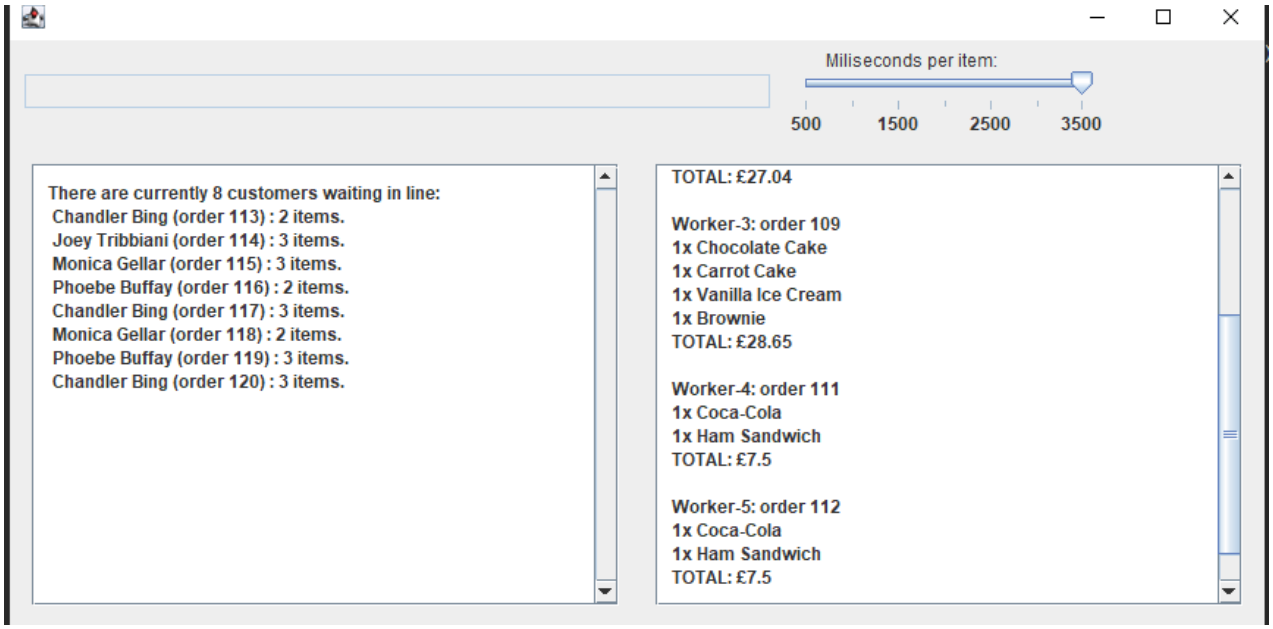


Figure 2: GUI set to slow; additional worker hired

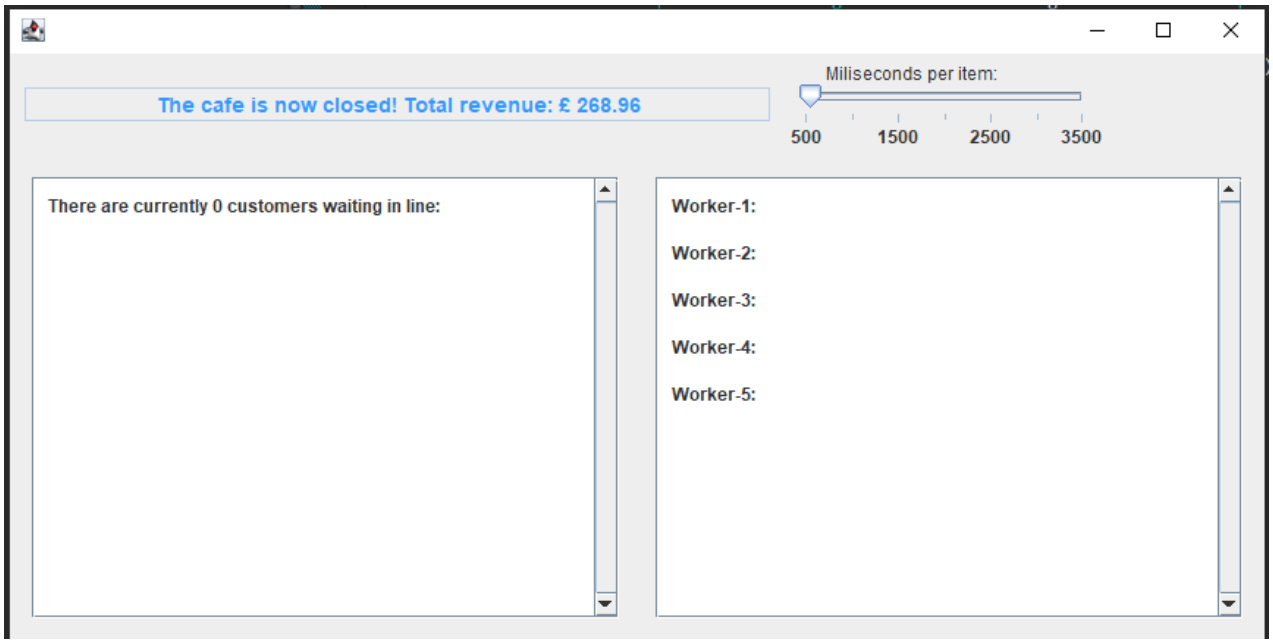


Figure 3: GUI finished; shows total revenue

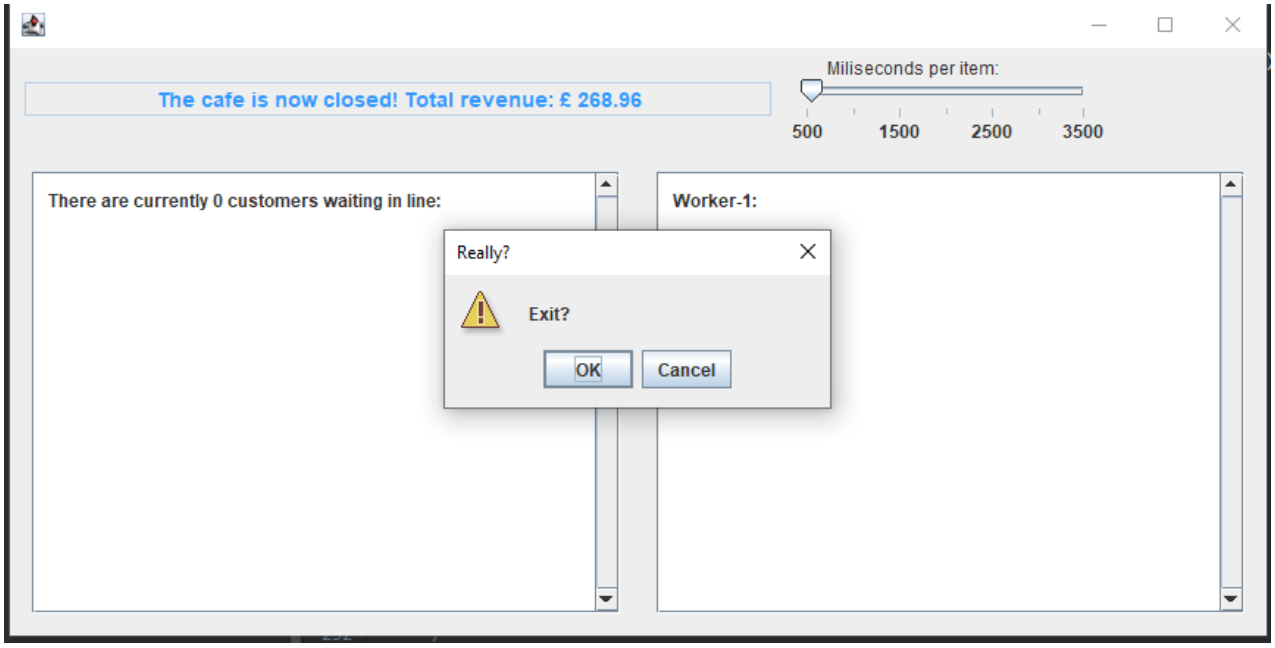


Figure 4: Upon pressing 'x', you are asked if you really want to exit