Design Document

Conor Kostick, Vincent Achukwu

Link to Demo Video

https://youtu.be/AZvXaIwxIGc

Introduction

In this document, we will break down how we divided the work between us, why we went with our approach to the car painting automation problem, and briefly explain each part of the code.

To run our program, simply execute the java file in a terminal by doing the following:

```
javac CarPaintingAutomation.java
java CarPaintingAutomation
```

Division of work

Upon beginning this assignment, we both broke down the project specification into smaller problems and tried identifying any patterns/threading problems that we could recognise from the problem statement. For instance, we observed that perhaps the way the interaction between the warehouse and the dealership was described may be similar to the producer/consumer problem - the dealership is the consumer (ordering cars based on what the customer selected) and the warehouse is the producer (producing cars at certain intervals). The buffer, in this case, would be the trailer capacity, meaning, the cars that were ordered will only ship to the dealership once the trailer capacity is full and only then will it trigger the delivery to the dealership.

Afterwards, Vincent wrote a general producer/consumer Java program after which we expanded it to better suit our car automation program functionality. As well as that, Vincent mainly focused on:

- the initial simple interaction between the warehouse and the dealership with a small number of models stored in a list of Car objects
- a simple painting functionality (a random car was selected and would be "painted" via a print statement stating that it's now being painted). Initially, this did not have any steps such as washing, electro dipping, etc.
- Specifying Car and Customer attributes (code was improved by replacing the Customer class with a Dealership class by Conor)

Conor further enhanced the car automation program by refining what has been implemented so far as well as developing the following:

- Making each painting process a thread (essentially each painting step is a robot)
- Implementing a config file mechanism where the user running the program could configure the parameters of the program (e.g dealership count, model count, number of production lines (which we called “maxPools”, etc)). This can be found in the config.txt file
- Output.dat file which is used for storing the program output log. This file is overwritten every time the program is executed.

The overall process of breaking down this assignment involved visualising how the program should run and understanding why we decided to use the classes in our code.

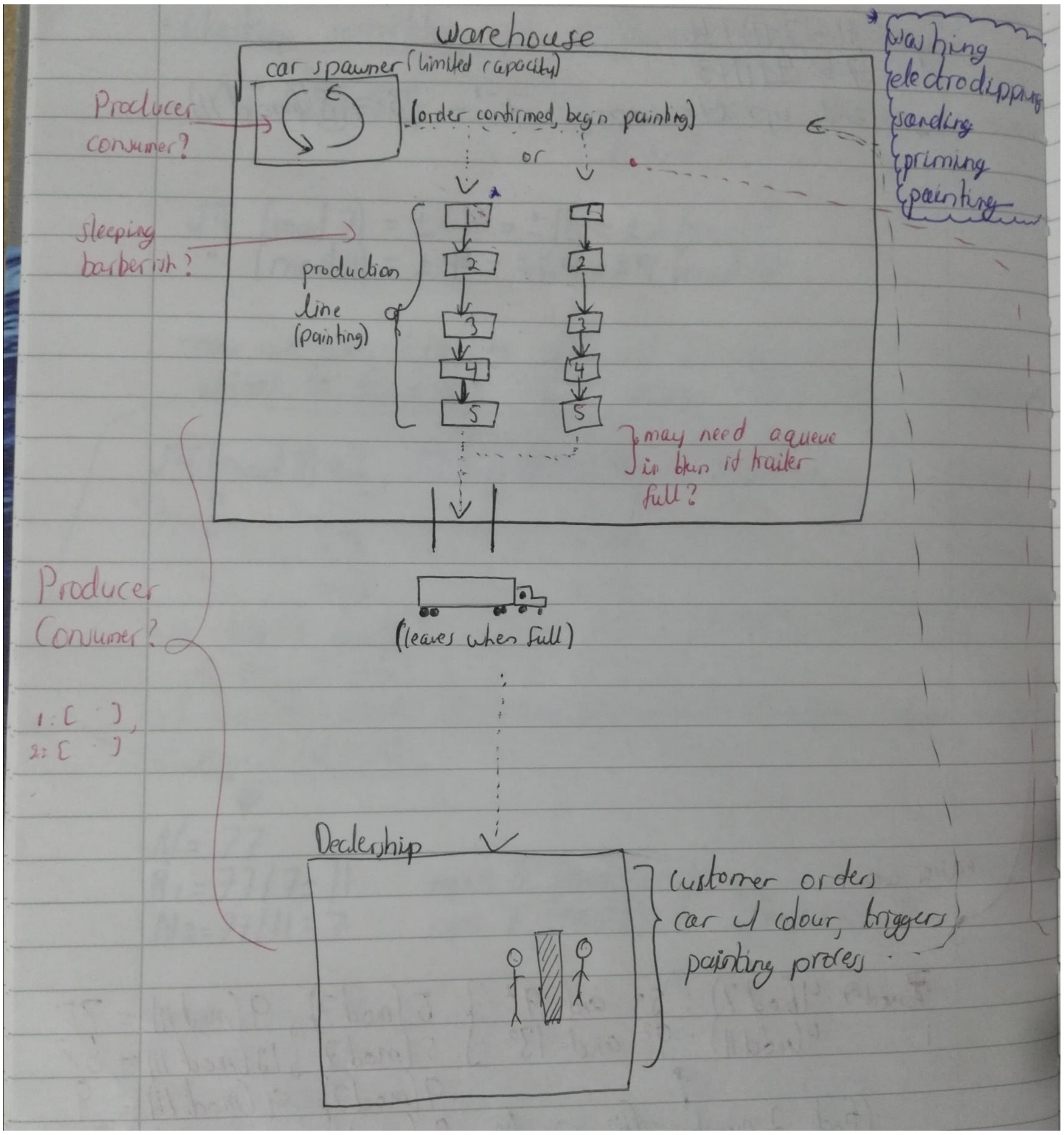
General Program Description

Initially, the program reads in the configurable parameters from the config file (config.txt). The general idea of our implementation is to have the producer produce cars every hour as long as the number of cars does not exceed the warehouse capacity. While those cars are produced, the warehouse notifies the consumer (i.e the dealerships) that there are cars available to order. A random dealership will order a random car model of a random colour. Once this happens, the car enters the painting process and goes through 5 steps: washing, electro dipping, sanding, priming, and painting. Each step has a default time for how long it takes to complete, and we were aware that it must take a certain amount of time per step depending on the car model. We determined the time taken per step by simply adding the price of the car ordered by the dealership to the default time on the current step the car is in. While a selected car is undergoing its painting process, the warehouse can produce more cars and the consumer/dealership can consume/order cars.

Once a car has completed its painting process and undergone each step, it is loaded to the trailer but it will not be shipped to its designated dealer until that dealership’s trailer capacity is full (given that each dealership has its own trailer for delivering the car models in their selected colour).

While all of this is happening, we have print statements to help better understand what stages the car painting automation is undergoing, while also logging those outputs to a dat file (output.dat).

The diagram below shows a general idea we initially had of how a dealership and the warehouse may interact with each other along with the painting process.


warehouse
car spawner (limited capacity)
Producer consumer?
(order confirmed, begin painting)
or
washing
electrodipping
sanding
priming
painting
sleeping barberish?
production line (painting)
1
2
3
4
5
may need a queue in bin if trailer full?
(leaves when full)
Producer Consumer?
1: [ ],
2: [ ]
Dealership
(customer orders car w/ colour, triggers painting process

Addressing Potential Issues

- We recognised that the producer / consumer style situation of the warehouse / dealership was independent to the painting of the cars, so we were able to make these two distinct sections run in parallel
- We dealt with the problem of fairness (in terms of which car gets painted next) by simply picking the car that has been waiting the longest to get painted
  - This approach also ensures that there is no starvation as every car will eventually get its turn to be painted
- We considered it unlikely to have any problems with deadlock since the order in which threads are waiting on each other is fairly linear.

Limitations of our implementation

Although our program demonstrates the process of how cars are produced, ordered, and painted, there are some limitations of our implementation.

- The config file could be better and improved with more configurable parameters. It also could have been stored in a neater format such as JSON which would have likely made it easier to add more detail.
- Only one car can be painted in a production line at a time. The issue here is that if, for instance, a car in production line 1 has finished washing and is now in the electro dipping process, another car cannot enter the washing process of production line 1 even though that stage of painting is now free and unused. Once a car has fully passed through all steps of the production line (washing, electro dipping, sanding, priming, and painting), only then can another car that has been ordered enter that production line (unless another production line is free). The maxPools parameter in the configuration file is also known as the maximum number of production lines in the warehouse.
- Customers can only order vehicles that have already been made. Rather than having a customer only ordering vehicles that are in stock, it would have been a better design approach to allow customers to order vehicles that have not yet been produced (i.e this would be equivalent to preordering a vehicle which would then later be produced and stored in the warehouse).
- The execution of our program may be hard to follow through the dat file and the console output. Therefore, a UI implementation may have been useful for better visualisation of the processes involved throughout the program execution.