The Factory Problem

Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s): Eoin Clayton, Robin O'Shea
Programme: CASE 4
Module Code: CA4006
Assignment Title: The Factory Problem
Submission Date: 22/03/2020
Module Coordinator: Rob Brennan

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at http://www.dcu.ie/info/regulations/plagiarism.shtml ,

https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment

guidelines.

Name(s): Eoin Clayton, Robin O'Shea Date: 22/03/20

Overview of the system

The primary goal of this assignment is to deliver a concurrent multithreading system and doing so in one's own interpretation. Conceptually the system is designed to facilitate a production line (thread pool) of robots (Threads) for installation of parts needed for and individual aircraft. The system is designed to deal with the management of decision making in threading assignment.
The system implements four main classes that are Aircraft, Production Line, Robot and storage. Each of which is integral to the next.

The system starts running from the main function in the Aircraft class. It first creates ten aircrafts and utilizing the method getRandomNumberInRange(int min, int max) each plane is instantiated at a random time interval. These aircrafts are assigned and ID eg.1, and creates a random subset of parts needed for installation from the following list of parts:
("A+","B+","C+","D+","E+","F+","G+",H+","I+","J+").

```
Aircraft(Aircraft_1 [E+|])
Aircraft(Aircraft_2 [B+|I+|D+|])
Aircraft(Aircraft_3 [H+|F+|B+|C+|J+|G+|D+|I+|E+|])
Aircraft(Aircraft_4 [E+|A+|C+|G+|H+|J+|])
Aircraft(Aircraft_5 [J+|E+|A+|F+|])
Aircraft(Aircraft_6 [D+|G+|J+|])
Aircraft(Aircraft_7 [E+|A+|I+|J+|F+|G+|D+|B+|])
Aircraft(Aircraft_8 [B+|E+|C+|H+|A+|J+|G+|F+|D+|I+|])
Aircraft(Aircraft_9 [D+|I+|B+|])
Aircraft(Aircraft_10 [E+|D+|I+|B+|H+|A+|])
```

The aircraft ID along with its list of requested parts is then passed into the production line. Once in the production line, the list of parts needed is broken up into individual tasks as each robot has a specific part to work on. The production line creates a new thread which calls the robot class for each task to be carried out.

```
final ExecutorService Robots = Executors.newFixedThreadPool(NumberOfRobots);

for (final Integer key : Robot.RobotParts.keySet()) {

    if (Robot.RobotParts.get(key).equals(partNeeded)) {
        robotNeeded = key;
        System.out.print("Robot " + robotNeeded + " needed \n");

        Robots.execute(new Robot(robotNeeded, id));

        Robots.shutdown();
```

In the robot class a check is carried out to see if the robot has already been assigned or is ready to carry out work. If a robot is already assigned the task waits until the robot has finished the current task before being executed. If the necessary robot is free it gets

assigned to the aircrafts tasks. After a fixed time the robot finishes the task and is then free to complete a new task. Once all tasks for an aircraft is complete, that aircraft is released and installation on a new aircraft begins.

A pool of total resources for all robots to use is also tracked. Once this pool reaches five remaining parts a single thread is created which calls the storage class to place an order to restock the pool with new parts. This order executes independently of the threads for the robots. The robots can keep carrying out their tasks and just as the pool of resources is empty, the order is completed.
Once all aircrafts requests have been completed the production line terminates.

```
public synchronized void orderParts(int numberOfParts){
    if(Robot.numberOfParts == 5){
        //System.out.println("Hello");
        try {
            System.out.println("Ordering new parts");
            TimeUnit.SECONDS.sleep(5);
            Robot.numberOfParts = 20;
            System.out.println("Order arrived. Number of parts remaining: " + Robot.numberOfParts);
        } catch (Exception e) {
            e.printStackTrace();

        }
```

How to compile and run

Download all files and ensure they are all in the same folder.

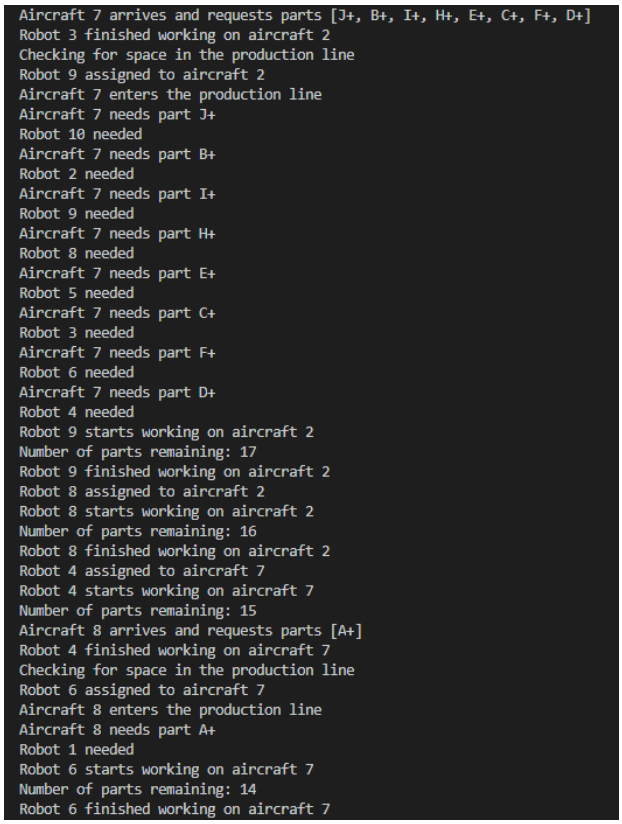
To compile the files run javac *.java in the terminal

```
C:\Users\eoinc\Documents\concurrent>javac *.java
```

Once compiled run java Aircraft to start the program

```
C:\Users\eoinc\Documents\concurrent>java Aircraft
====Start====
Aircraft 1 arrives and requests parts [J+, G+]
Checking for space in the production line
Aircraft 1 enters the production line
Aircraft 1 needs part J+
Robot 10 needed
```

Results



The above image is a sample output from running the program. As you can see Aircraft 7 arrives and requests parts [J+, B+, I+, H+, E+, C+, F+, D+].
The relevant robot is then found. Once aircraft 2 has finished the robots get assigned to aircraft 7 and the number of parts remaining decreases by 1.
Half way through aircraft 8 arrives and requests part [A+].

Addresses issues

We decided that the best way to address the issue of fairness is to assign the aircraft which arrives first to enter the production line and have its necessary parts installed first. If two aircrafts arrive at the same time, the aircraft with fewer parts required will be seen to first.

To prevent starvation the production line will not sleep until all requests are complete and all aircrafts have left the production line. The total number of parts available will also restock using the just in time method. The production line can then go to sleep while waiting for a new request.

Dividing the work

We started with both of us working on separate classes which were the aircraft and production line classes. We then decided to work on the robot and storage class together by completing separate functions and communicating together to ensure the system was behaving the way we intended.