

## Declaration on Plagiarism

### Assignment Submission Form

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

Name(s): Matthew Nolan, Michael O'Hara
Programme: CASE 4
Module Code: CA4006
Assignment Title: Concurrency Assignment 1
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.

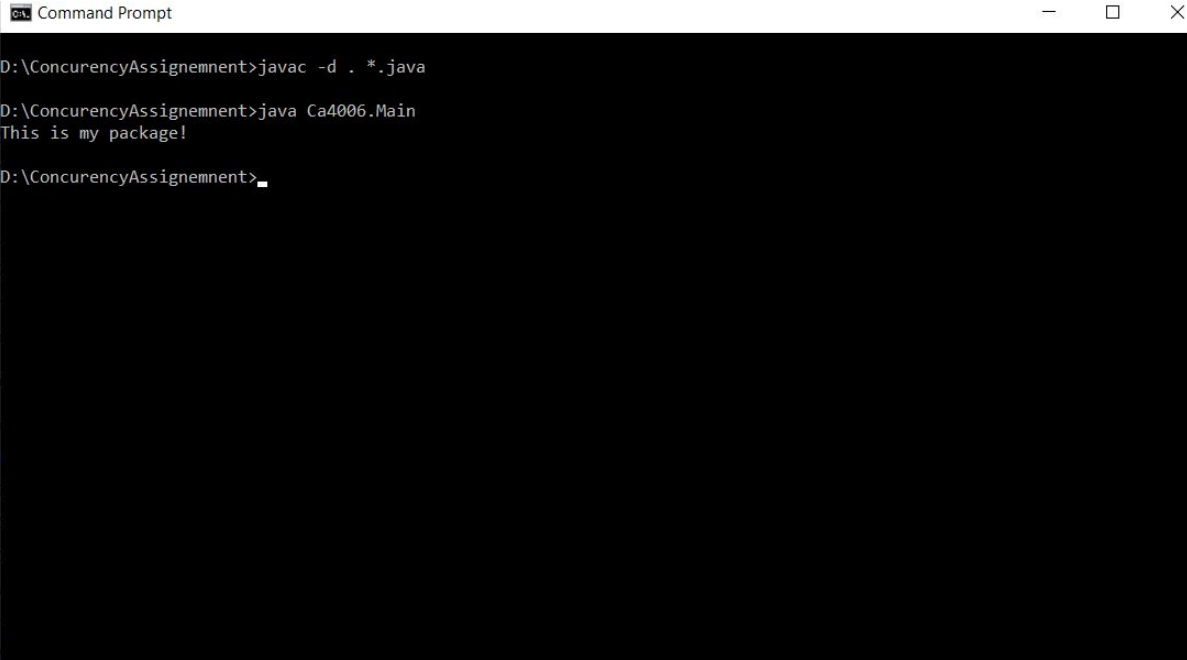
Name(s): Michael O'Hara, Matthew Nolan  
Date: 18/03/2020

# CA4006 Assignment 1

## How to compile & run:

The first command in the screenshot below is to compile all the java files in the directory.

The second command in the below screenshot is used to run the Package CA4006 which will run our assignment and print out the log for the robots and aircrafts.



```
Command Prompt
D:\ConcurrencyAssignment>javac -d . *.java
D:\ConcurrencyAssignment>java Ca4006.Main
This is my package!
D:\ConcurrencyAssignment>
```

## Classes and Setup:

For our implementation we have 9 secondary classes and one Main in our package. The 5 classes we have in our implementation are as follows:

- Aircraft
- Aircraft Controller
- Aircraft Waiting List
- InProductionList
- Part
- Production Line
- Storage
- Robot
- Robot Controller

The class called Main in our implementation is responsible for starting our production line which is going to be our primary thread. The main thread is responsible for starting the robot

controller ,the aircraft controller and production line. The Robot Controller is also responsible for initiating the threadpool. (A thread pool is a group pre-instantiated idle threads which are ready to be given a task to complete). This was used to avoid having to instantiate new threads each time. The aircraft controller is then responsible for creating new aircraft which will need to have some amount of work done on them. Once a new aircraft has been made it gets into the queue of Aircraft which are waiting for parts. At the same time it will notify the production that there is an aircraft waiting to be worked on. The aircraft controller will check the storage facility to see if the parts needed for the aircraft are available to be used.

The aircraft is then added to the production line, once this occurs the robot controller is notified. The robot controller checks if there is a robot free. If there are no free robots then the aircraft must wait, if there is a free robot then said robot is assigned to the aircraft. The robot begins working on installing the parts. This is executed in the thread pool.

## Self Reflection:

### Contribution to Project

#### **Michael:**

- Robot
- RobotController
- Part
- Aircraft
- Storage

#### **Matthew:**

- InProductionLine
- ProductionLine
- AircraftController
- AircraftWaitingList
- Main

### Lessons Learned

The first thing we learned was that implementing threads and concurrency was very different in Java than it is in other languages we have become familiar with such as Python. We had not used Thread pools and synchronised functions in practice before starting work on this assignment. This meant before we could get to work on the code we had to familiarize ourselves with how they worked in practice. It took us some time to figure out the intricacies of these we were able to put them to use.

The use of a thread pool allowed us to set a set of tasks to be completed and pass them to the executor service. This allows the runnables that are to be run by a thread to be run sequentially.

Once we began working our implementation we realised that our initial idea was somewhat flawed so we had to make minor changes to it.

We decided it would be best to add in an Aircraft Controller and a Robot Controller in order to control the jobs related to Aircraft and Robot respectively.

The Aircraft Controller would be responsible for making new aircrafts and adding those aircraft into a waiting list before being put into production by our waiting list.

Robot Controller is responsible for waiting free space in the production line and once an aircraft moves into the production line, the Robot Controller will then assign a Robot to the aircraft which has been added to the Production Line.

We felt that this was the best approach as the Controllers and the Production line act as our three main threads. These will interact with the shared resources (Waiting List and In Production List). The Production Line interacts with both of the shared resources, The Aircraft Controller will interact with the Waiting List and the Robot Controller will interact with the In Production List.

We felt that this gave us more control over what each thread was in charge of. It was preferable to the initial design, which gave all control to the storage but this didn't give us the result we wanted so we opted for this approach. Our old design didn't allow for the level of communication between the threads that we needed for the robots to be able to work on the aircraft.

## Issues:

We also had some issues with the code, these included interprocess communication, getting threads to end once the function had finished and the use of notify/wait statements. In our system we implemented two shared resources, the AircraftWaitingList and the InProductionList caused some issues with synchronised functions. Our main issue was getting the threads to close once they had finished their purpose. We realised that the wait() in InProductionLine was causing the threads to hang once there were no more aircrafts being made. We were unable to come up with a solution to this problem.