Declaration on Plagiarism

Assignment Submission Form

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

Name(s):	Luke Hebblethwaite(17425212), Jack Doherty(17351591)
Programme:	CASE4
Module Code:	CA4006
Assignment Title:	Programming Assignment 1
Submission Date:	19/03/2021 17:00
Module Coordinator:	Rob Brennan

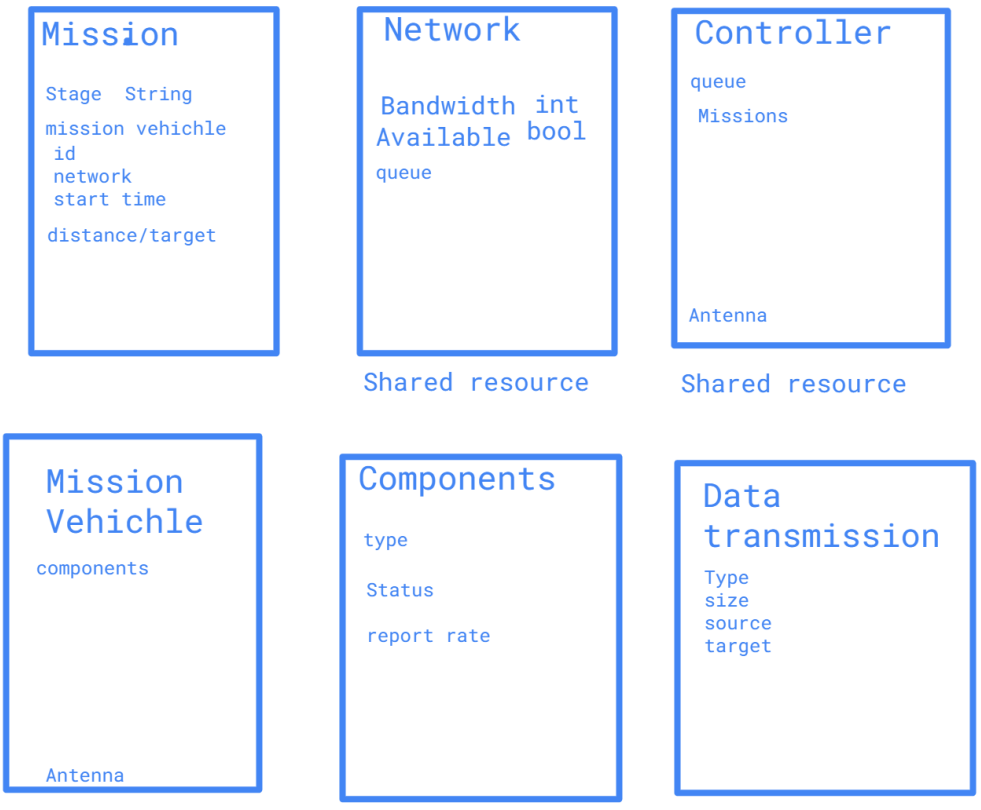
I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying 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): _Luke Hebblethwaite, Jack Doherty_____Date: ___19/03/2021_____

Design Overview

When we were first assigned this problem we decided on an approach represented by the class diagrams below which included a class for Mission, Network, Controller, Mission Vehicle, Component and Data Transmission.



Stages chance of failure: 10%
25% of failures can be recored by sending software upgrades - var days var size mb
30% of reports require a command response- mission paused intil comand recieved
variable burst of reports and commands are sent at the transition between mission stages
reports - telemetry = 100-10k bytes frequent or data 100k-100mb periodic

Upon starting to implement the above design we realised we were over complicating the design in some areas such as the Mission Vehicle and Data Transmission class so we quickly dropped these classes and represented the data transmissions simply as strings and contained the mission components inside the Mission class rather than adding the extra layer that was the Mission Vehicle class. We also changed our planned implementation of the networks. We decided to represent them as a concurrent hashmap with strings mapped to NetworkQueues. We had to make another class called NetworkQueue that simply contained a String queue.

Main.java
This is our main class. In this class we scan for input, this input being the number of random missions to generate. When we generate a mission we randomly select a mission destination and an associated distance. From this distance the number of components for the mission is calculated e.g. Fuel required is the distance of the mission in millions of kilometres, divided by 2 in kilograms of fuel. This class then creates the thread pool of size 100, submits the missions and mission controller to it as tasks and calls it to execute. This class also initialises the Network class as a ConcurrentHashMap. As it is a shared resource between both the missions and mission controller it is included in their construction.

MissionController.java
This class contains all of our Mission Controller logic. The Mission Controller waits for messages from the missions via the controller network, if the response is not null, the response items are noted. Depending on the message received, different functions are executed. For example, if the message is incremented, the controller responds to the mission with a positive approval for the mission to increment itself. If the message is a stage failure the controller checks if the failure can be recovered from via a software upgrade. If this is the case, the controller generates a software upgrade of random size (25-50 MB) and sends this to the mission via the large network. If this message is a component report the controller logs the report and responds to 30% of these reports with an acknowledgement message. Finally, if the message is “ended” the controller logs that a mission has ended and ends itself when all missions have completed.

Network.java
This class uses a ConcurrentHashMap to represent our networks. It maps three strings “2MB”, “2KB” and “2B” representing the 3 different network bandwidths. It does this for both the controller direction and mission direction. It then adds those types of reports to the network queue. This class uses our NetworkQueue class to create our network queue.

For example:
Network instance points to :
{
	“2MB Contoller” : NetworkQueue instance
	“2KB Contoller” : NetworkQueue instance
	“2B Controller” : NetworkQueue instance
	“2MB Mission” : NetworkQueue instance
	“2KB Mission” : NetworkQueue instance
	“2B Mission” : NetworkQueue instance
}

Mission threads add onto the Controller type NetworkQueues which the Controller looks to for messages and the Controller adds onto the Mission type Queues which the missions look to for messages.

Unfortunately we have not implemented simulating the speed of the different networks though we had planned to do this by calculating the number of seconds a transmission would take and calling the network to sleep for this number of seconds. For example, if a software upgrade of size 23Mb were to be sent via the 2Mb network would be called to sleep

for 12 seconds before adding the String to the Queue to simulate the time it would take for this message to transmit.

NetworkQueue.java
This class defines the individual network queues as String queues with associated methods to add remove etc from the queue.

Mission.java
This is our mission class. Its constructor takes in a number of parameters that relate to the mission details and also its components as component objects. All components are then started within the constructor. The run() method handles all running logic of the mission, such as sending messages outlining whether or not the increment stage was successfully completed, checking for component reports logging etc.

Component.java
This class's constructor takes in parameters such as missionID, type of component and status of component. The run() method starts the component. Each component also has a random report rate between 15-30 seconds. In this way a component with a report rate of 17 generates a report every 17 seconds. The mission class checks each of its components for reports and if they have a report generated at the time of the check that report is sent to the mission controller via a network for the report to be logged to output by the controller. This class also has a report() method which returns the details of the component at the time it is called.

Concurrency Problems
The main concurrency problem we identified was the issue of the shared resource in the form of the networks. To handle this problem we utilized a ConcurrentHashMap to represent the networks which is thread safe i.e. multiple threads can operate on a single object without any complications. This means at a time any number of threads can perform retrieval operations but for updation in the object the thread must lock the particular segment in which the thread wants to operate. This type of locking mechanism is known as segment locking or bucket locking. Hence at a time, 16 update operations can be performed by threads.

Another concurrency problem we identified was starvation and fairness. Potentially we could see a scenario where a mission is favoured over others and is allowed to increment again and again without other missions being allowed to increment their stage. Additionally if a scenario occurs in which there are a lot of components generating reports at the same time this could create scenarios in which the missions messages to be incremented are never heard by the controller and so the missions never end and the program runs indefinitely. To tackle this we decided to to represent the individual networks as queues rather than our initial approach of a single direct communication channel. In this way all mission calls to be incremented are eventually seen by the controller even if there is a large number of component reports or other missions requesting to be incremented.

https://www.geeksforgeeks.org/concurrenthashmap-in-java/

Work Division
Each came up with initial designs and assumptions about the assignment then met and discussed and decided on an approach. Jack took the main role of developing the code and we pair programmed portions of the code when this was an option. We communicated with each other throughout the development of the code and made a group decision on any new updates and changes to the design and implementation.

How To Run
How to compile and run the code:

```
C:\Concurrent\assignment_1>javac assignment_1/Main.java

C:\Concurrent\assignment_1>java assignment_1/Main
```

```
C:\Concurrent\assignment_1_luke>java assignment_1/Main

Enter amount of missions: 10
```

Jack Doherty personal reflection
From this assignment I learned a lot about the concepts of fairness and starvation in relation to our implementation of the networks and also a good deal about the java concurrency utilities such as some of the objects we made use like thread pools and concurrent hashmaps. I think it relates very well to the course material and allows us freedom to learn more about concurrency programming ourselves within the scope of java concurrent programming. Next time I would be much more mindful of time and the amount of time it would take to decide on a final implementation design and then the time it would take to implement that design.

Luke Hebblethwaite personal reflection
From this assignment, I feel that I learned a lot about the concept of concurrency in general. I feel that I learnt about the contents of this module in a practical way and that has helped my understanding massively. I had never used a thread pool before and this is something I would definitely use again in the future. I also felt that this being an assignment done in teams helped a lot as Jack and I were able to pair program and help each other with aspects that the other person didn't understand. If I were to start this project again, I would consider using other concurrent programming concepts as there may be better suited implementations to this problem.
Overall, I enjoyed this assignment as it has solidified a lot of concepts and look forward to the next one.