INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT074-3-2-CCP

CONCURRENT PROGRAMMING

UC2F1908CS(DA) /UC2F1908CS

HAND OUT DATE: 24 MARCH 2020

HAND IN DATE: 22 MAY 2020

WEIGHTAGE: 25%

INSTRUCTIONS TO CANDIDATES:

1 Submit your assignment at the administrative counter

2 Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing)

3 Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld

4 Cases of plagiarism will be penalized

5 The assignment should be bound in an appropriate style (comb bound or stapled).

6 Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.

7 You must obtain 50% overall to pass this module.

INDIVIDUAL ASSIGNMENT

(PART 2)

CT074-3-2-CCP

CONCURRENT PROGRAMMING

UC2F1908 CS(DA)

Name: Chan Jia Le (TP049952)

Proposal Title: Concurrent Programming

Hand Out Date: 24th MARCH 2020

Hand in Date: 22nd MAY 2020

Table of Contents

1.0 | Introduction

Upon the completion of the simulation, most the assumption made in the first report remains true except for some part of the implementation. In the first report, it is assumed that customer might leave the queue if waiting time is too long, but now the assumption has been modified and the customer that stays on the queue will wait until they have the chance to enter, unless it is almost closing time and owner has shouted last call. If owner had called "last call", the customer that is queueing will be forced to leave.

2.0 | Different Concurrent Programming Facility

2.1 | Linked List and Array List

```
LinkedList<Customer> listCustomer = new LinkedList<Customer>(); //list of customers
public final List<Customer> allCustomer = new ArrayList<>(); //array list to contain all the customers
```

Figure 2.1.1 Linked List and Array List sample

Figure 2.1.1 shows the code snippets for linked list and Array List. Linked list and array list are implemented for store the customer that is generated. Once the customer is generated it will then be stored inside the linked list and array list for getting generated customer for better management.

3.0 | List of Requirement Met

3.1 | Owner and Waiter

```
Thread-Customer 1 : attempt getting in Cafe.
Thread-Customer 2 : attempt getting in Cafe.
Thread-Customer 1 : has entered the Cafe.
Available Seats :9
Thread-Waiter : is now taking order from Thread-Customer 1 :
Thread-Customer 1 : had ordered Fruit Juice
Thread-Waiter : taking glass from Cupboard
Thread-Customer 2 : has entered the Cafe.
Available Seats :8
Thread-Owner : is now taking order from Thread-Customer 2 :
Thread-Customer 2 : had ordered Cappucino
Thread-Waiter : is now making Fruit Juice
Thread-Owner : taking cup from Cupboard
Thread-Owner : taking milk from Cupboard
Thread-Owner : taking coffee from Cupboard
Thread-Customer 3 : attempt getting in Cafe.
Thread-Owner : is now making Cappucino
Thread-Waiter : is now using Juice Tap.
Thread-Customer 3 : has entered the Cafe.
Available Seats :7
Thread-Waiter : had finished making Fruit Juice and served to Thread-Customer 1 :
Thread-Owner : had finished making Cappucino and served to Thread-Customer 2 :
```

Figure 3.1.1 Owner and Waiter taking orders independently

Figure 3.1.1 shows that when the customer had entered, both the waiter and owner will work together to take orders from different customer. As shown in the figure, when customer 1 entered thread-owner takes the order from the customer 1 and proceed to complete the order, while thread-waiter is serving customer 2.

```
----------------IT IS ALMOST CLOSING TIME----------------
Thread-Owner :  LAST CALL!!!
```

Figure 3.1.2 Owner calling for last call

While closing time is coming soon, last call will be called by the owner.

```
Thread-Owner : is now taking order from Thread-Customer 1 :
Thread-Customer 1 : had ordered Fruit Juice
Thread-Owner : taking glass from Cupboard
Thread-Owner : is now making Fruit Juice
Thread-Owner : is now using Juice Tap.
Thread-Owner : had finished making Fruit Juice and served to Thread-Customer 1 :
```

Figure 3.1.3 Owner serving customer fruit juice

Figure 3.1.3 shows the owner taking order and creating fruit juice for customer.

```
Thread-Waiter : is now taking order from Thread-Customer 1 :
Thread-Customer 1 : had ordered Cappucino
Thread-Waiter : taking cup from Cupboard
Thread-Waiter : taking milk from Cupboard
Thread-Waiter : taking coffee from Cupboard
Thread-Waiter : is now making Cappucino
Thread-Waiter : had finished making Cappucino and served to Thread-Customer 1 :
Thread-Waiter : returning ingredients back to cupboard
```

Figure 3.1.4 Waiter serving customer cappuccino

Figure 3.1.4 shows the waiter taking order and creating cappuccino for customer.

3.2 | Customer

```
Thread-Customer 1 : attempt getting in Cafe.
Thread-Customer 1 : has entered the Cafe.
Available Seats :9
Thread-Waiter : is now taking order from Thread-Customer 1 :
Thread-Customer 1 : had ordered Cappucino
```

Figure 3.2.1 Customer enters the café and proceed to waiter proceeds to take order from customer

```
Thread-Customer 2 : attempt getting in Cafe.
Thread-Customer 2 : has entered the Cafe.
Available Seats :8
Thread-Owner : is now taking order from Thread-Customer 2 :
```

Figure 3.2.2 Customer enters the café and proceed to owner proceeds to take order from customer

Figure 3.2.1 and figure 3.2.2 shows customer entering café and order, once customer enters available seat will be printed.

3.3 | Table and Seats

```
Thread-Customer 13 : waiting in queue.
Thread-Owner : taking milk from Cupboard
Thread-Customer 14 : attempt getting in Cafe.
Thread-Owner : taking coffee from Cupboard
Thread-Customer 14 : waiting in queue.
```

Figure 3.3.1 Table fully occupied

There are only 10 seats in one table, and once it is full, the customers will not be allowed to enter café and must queue.

3.4 | The Clock

```
----------------IT IS ALMOST CLOSING TIME---------------
Thread-Owner :  LAST CALL!!!
```

Figure 3.4.1 Clock Notify Owner

```
----------------IT IS ALMOST CLOSING TIME---------------
Thread-Owner :  LAST CALL!!!
Thread-Waiter : will now leave the cafe
```

Figure 3.4.2 Clock Notify Owner

```
----------------IT IS ALMOST CLOSING TIME---------------
Thread-Waiter : taking coffee from Cupboard
Thread-Waiter : is now making Cappucino
Thread-Owner : LAST CALL!!!
Thread-Owner : is now taking order from Thread-Customer 14 :
Thread-Customer 14 : had ordered Fruit Juice
Thread-Owner : taking glass from Cupboard
Thread-Customer 19 : are forced to leave because cafe is closing
Thread-Customer 20 : are forced to leave because cafe is closing
```

Figure 3.4.3 Clock Notify Owner and Forcing Customer to leave

The clock will notify owner and waiter when almost closing, waiter will finish the current job and leave the café as assumed in first report. Customer will be forced to leave when last call is called by the owner.

```
------------------CAFE IS NOW CLOSED !--------------------
```

Figure 3.4.4 Clock notify closing time

Notifying closing time by clock.

3.5 | Statistics

```
--------------------STATISTICS----------------------
Total amount of fruit juice ordered : 7
Total amount of coffee ordered      : 11
Total amount of customer served     : 18
Total amount of customer left       : 2
Thread-Owner : will now leave the Cafe
```

Figure 3.5.1 Statistics Printed before owner leaves

Statistics printed before owner leaves café.

3.6 | Configuration

```
int numCust = 20; // number of customer desired to be generated
```

Figure 3.6.1 modifiable number of customers generated

Number of customers generated can be configured.

```
public synchronized void useJuiceTap(String position) throws InterruptedException{
    System.out.println(position + " is now using Juice Tap.");
    Thread.sleep(1000); //time to pour fruit juice, can be modified
}
```

Figure 3.6.2 modifiable juice making time

```
System.out.println(position + " is now making Cappucino");
//wait time for mixing cappucino
Thread.sleep(1000); //can be modified
```

Figure 3.6.3 modifiable juice making time

Configurable time for pouring juice and mixing coffee.

4.0 | Concurrency Concept Implemented

The concepts that are implemented are like the ones from the first report except for the exclusion of semaphore. Sleep function is used to simulate time, atomic statement is implemented to avoid erroneous counter results and synchronized being implemented to avoid unwanted output when resources is shared as it will lock the resources and release once the thread has finish executing the action.

4.1 | Time Simulation

```
public void takeCup(String position) throws InterruptedException{
    System.out.println(position + " taking cup from Cupboard");
    Thread.sleep(500);
}
```

Figure 4.1.1 Time Simulation using Sleep

4.2 | Atomic Statements

```
AtomicInteger juiceOrder = new AtomicInteger(); //total fruit juice ordered
AtomicInteger coffeeOrder = new AtomicInteger(); //total coffee ordered
AtomicInteger customerServed = new AtomicInteger(); //total customer served
AtomicInteger totalCustomer = new AtomicInteger(); //total customer left because of cafe closed
```

Figure 4.2.1 Atomic Statement Variable initialization

```
//function to increment cappucino order
public synchronized void addCustomerServed(){
    customerServed.getAndAdd(1);
}
```

Figure 4.2.2 Atomic Statement for counting customer served

4.3 | Synchronizations

```
//using juicetap to pour juice and create drink
public synchronized void useJuiceTap(String position) throws InterruptedException{
    System.out.println(position + " is now using Juice Tap.");
    Thread.sleep(1000); //time to pour fruit juice, can be modified
}
```

Figure 4.3.1 Synchronization for shared resource