Objective:

This assignment has been designed for students to apply appropriate concurrent program methods in implementing a concurrent program from a program specification.

Learning Outcomes

ON COMPLETION OF THIS ASSIGNMENT YOU SHOULD BE ABLE TO DEMONSTRATE THE FOLLOWING LEARNING OUTCOME(S):

No.	Learning Outcome	Assessment
1	Explain the fundamental concepts of concurrency and parallelism in the design of a concurrent system (C2, PLO1)	Exam
2	**Apply the concepts of concurrency and parallelism in the construction of a system using a suitable programming language. (C3, PLO2)**	**Individual Assignment (System)**
3	Explain the safety aspects of multi-threaded and parallel systems (A3, PLO6)	Individual Assignment (Report)

Programme Outcomes (PO):

PLO2 Cognitive Skills - This relates to thinking or intellectual capabilities and the ability to apply knowledge and skills. The capacity to develop levels of intellectual skills progressively begins from understanding, critical/creative thinking, assessment, and applying, analysing, problem solving as well as synthesizing to create new ideas, solutions, strategies or new practices. Such intellectual skills enable the learner to search and comprehend new information from different fields of knowledge and practices.

Individual Assignment - System (25%):

Question No.	Topic	Question Vs Taxonomy						PLO
		Cognitive Level						
		1	2	3	4	5	6	
		SQ	SQ	SQ	SQ	SQ	SQ	
1	Appropriateness of coding techniques used to implement design with appropriate comment lines in source codes			**20%**				2
2	Appropriateness of the Java concurrent programming facilities used.			**20%**				2
3	Program runs appropriately with basic requirements			**20%**				2
4	Additional requirements met			**20%**				2
5	Explanations of concurrency concepts implemented with relevant code samples			**20%**				2
	Total			**100%**				

Submission Requirements

Assignment Handout Date : 24th March 2020

Assignment Due Date : 22nd May 2020

Case Study

The Problem

An owner of a cafe thinks that the way in which the cafe is operated means that customers have to spend too much time waiting to be served, and so tend to go elsewhere instead. In order to evaluate the situation a simulation of the Cafe has been commissioned. This simulation will simply run with text output describing the events as they occur in the Cafe, and collect a minimal amount of statistical data.

Intention of assignment

Even if valuable to the owner, the simulation is *not* the main purpose of this assignment - indeed, if so was the case there are much better techniques for *simulating* than writing a concurrent program.

Identify all potential sources of deadlock in the problem specification and describe briefly how they are avoided in the implementation. Your documentation should include at least the following:

- Could the Waiter and Owner be involved in a deadlock when they take the ingredients for the cupboard? Why (not)?
- Could the Waiter and Owner be involved in a deadlock when they take the glasses and cups for the cupboard? Why (not)?

The requirement of this assignment is to write a program in which synchronization and communication takes place between several concurrent processes. It is intended to force you to solve (and not simply avoid) a range of interesting synchronisation problems.

******************************Basic Requirements******************************

The Cafe

The cafe consists of a number of 1 table with 10 seats, a serving area containing a juice fountain tap and a cupboard (holding the clean glasses, cups, milk and coffee), a clock, an Owner, a Waiter, and a number of customers. The Restaurant is operated by the Owner and Waiter, whose job is to make and serve drinks to customers who order at the Bar in a first-come-first-served manner. Each customer may only order a single drink at one time. The Owner and Waiter can each serve only a single customer at a time, although others may be waiting to be served.

The Owner and Waiter

- There is one order queue to both the Owner and The Waiter.

- On receiving an order, the Owner or the Waiter will refuse to serve a drink for any customer ordering after closing time (although closing time is not *called* by the Owner in the traditional sense).
- If it is not closing time, the Owner and Waiter serve customers independently. They serve fruit juice or cappuccino, according to what the customer's order. After having received an order they go to the cupboard and take out a glass or a cup, according to the type of drink ordered. We can assume there is sufficient cups and glasses throughout the day.
    - Fruit juice: Obtain a glass (takes a set amount of time). Obtain the juice fountain tap. Fill the glass (also takes a set amount of time).
    - Cappuccino: Obtain a cup (takes a set amount of time). Obtain each ingredients coffee and milk (each take a set amount of time). Mix the drink (take a set amount of time).

  After making the drinks the ingredients are returned to the cupboard.
- The Waiter leaves at closing time. The Owner will not leave the restaurant until all others (customers and Waiter) have left the restaurant.

The Owner

- At ten-minutes before closing time the Owner calls *"last orders"* to warn the customers that closing time is soon.
- The Owner may be alerted by the Clock of this.
- The Owner may finish serving any customers order he started before attending to the *"last orders"* call.

The Serving area

The serving area consists of a fruit juice tap and a cupboard. The cupboard contains all the ingredients, and the cups and glasses.

- There are unlimited number of glasses and cups in the system.
- Each of the ingredients (coffee and milk) and the fruit juice tap are <u>separate</u> resources. Each of these resources may only be used by one person at a time.
- To serve a customer the Owner or the Waiter takes one glass or cup, collects all the ingredients they need, one at the time, makes and pours the drink, puts the ingredients back and finally serves the customer.

The Customers

- The customers enter the Cafe at regular time intervals spread throughout the lifetime of the simulation.
- On entering the Cafe, each customer will order a drink, which will be served by either the Owner or the Waiter.
- Each customer orders only one type of drink. The type of drink ordered is determined at random according to the set ratio.

****************************Additional Requirements****************************

The Tables

- There is only 1 table in the bar that seats 10 customers.
- The tables have unlimited number of units of space for cups and glasses.
- It must be possible to fill a table. If there are no free seats left on the table and a customer wants to enter they should not be allowed to enter the Cafe.

The Clock

The clock in the bar serves two purposes.

- It will alert the Owner of both *"last orders"* and *"closing time"*. The Owner will alert his customers when it is time for last orders.
- The clock serves and updates the current time that each process uses to print their actions. *This part is optional: processes can also use time stamps for statistics*.

The Statistics

At the end of the simulation, i.e. when all other processes have terminated cleanly, the Owner should do some *sanity checks* of the Bar and print out some statistics on the run. The result of the sanity checks must be printed. You must

- Check that all tables are indeed empty.
- Print out statistics on
  - Number of customers served.

Configuration

The following thing *must* be configurable at compile time:

- Owner/Waiter
  - Time to make cappuccino or pour fruit-juice.
- Customer
  - Number of customers entering the bar. It should be possible to set this to zero.
  - The ratio of different types of drinkers, cappuccino and, fruit-juice.

Sample Output

In order to see what is happening dynamically you must have output from the Customers, the Owner, the Waiter and the Clock reporting all their major events.

Add information about which process/thread is doing the output. This way you can see if a process/thread acts for another, which is strictly forbidden, but is a common error for Java solutions (objects are not processes!). An example of such incorrect behaviour is

```
Thread-Owner : 21.31: Owner: Peter is served!
main : 21.50: Owner: Last orders!
Thread-Customer-8 : 21.50: Kelly is going to order fruit-juice from Owner.
```

Where you can see that not only the Owner thread but also the main thread is acting for the Owner.

You *must not*

- Kill a thread or process. You may not use any of the following primitives in Java:
    - Thread.stop
    - Thread.resume
    - Thread.suspend
    - Thread.interrupt

  You may not use the destroy or stop(0) primitives in - except to take care of temporary resources like simple timers.

- Solve the last orders problem in a manner forbidden in the description above.

- Resolve communication with an all-purpose one-channel solution.

Implementation

You should implement your simulation in Java.

Documentation for System (Week 12)

The documentation should detail the system implementation and testing.

- *Introduction which states <u>any difference</u> in your assumptions <u>compared to Part 1</u> in the simulation.*
- *Justification of the Java concurrent programming facilities used <u>if different from Part 1</u>.*
- List of requirements met.
- Explanations of concurrency concepts (atomic statements, synchronization, etc) implemented with relevant code snippets.

500 words excluding references/appendix/coding.

Submission for System (Week 12)

- *Java files required to run the simulation*
- *Instructions to run the program.*
- *Video of the simulation running. Maximum 5 minutes.*
    - *One-day restaurant operation simulation in 30 seconds.*
        - 20 customers
    - Busy day restaurant simulation
        - 50 customers.
    - Disable the waiter.
        - 50 customers.

Marking Scheme

Criteria	**Total marks**	**Marks awarded**
Appropriateness of coding techniques used to implement design with appropriate comment lines in source codes	20	
Appropriateness of the Java concurrent programming facilities used.	20	
Program runs appropriately with basic requirements	20	
Additional requirements met	20	
Explanations of concurrency concepts implemented with relevant code samples	20	
TOTAL MARKS	100	