# TDT4186 Operating Systems

Assignment 1

February 2018

- **Delivery deadline: March 12, 2018 by 23:59**
- After the delivery deadline you have one more week to demonstrate your solution to a student assistant, by March 16. You may of course demonstrate your solution earlier if you finish before the deadline.
- This assignment is a mandatory exercise needed to be able to take the final exam.
- Cribbing from other students (koking) is not accepted, and if detected will lead to the assignment being failed.

# 1 Sushi Bar Simulation

## 1.1 The overall purpose:

In this assignment you are going to simulate a sushi bar. The implementation will need to manage *several threads* that are using *shared resources*. These resources should be protected using the monitor concept. The implementation should use the *producer/consumer model* to solve the problem. The Producer and Consumers should *synchronize* using the *wait( )*, *notify( )* and *notifyAll( )*, and *mutual exclusion* should be implemented using the Java *monitor*.

## 1.2 The Problem:

The sushi bar consists of a clock, a door, a waiting area and a number of waitresses. The bar is open for only a specific time period, in which customers can enter the waiting area. Because of the limited number of waitresses, customers may have to wait in the waiting area. When a waitress fetches a customer, he/she will order some sushi. Customers have the choice to eat some of their sushi in the bar and have the rest as takeaway.

When it is closing time, the bar cannot accept more customers and only those who have already entered the waiting area will be served (i.e. customers already in the waiting area are allowed to order).

## 1.3 The Customer:

When a customer is created it should be given a unique ID. Also, the customer should be able to order food, when he/she is fetched from the waiting area by a waitress. After ordering, the customer takes some time eating.

**Note** that the customer should have minimal functionality, leaving the adding and removing of customers in the waiting area to the producer (door) and consumers (waitresses).

## 1.4 The Door:

The Door will work as the producer.

The door creates customers at random intervals and tries to put them in the waiting area if there is room. If there is no more room in the waiting area the door should wait until there is room for a new customer.

## 1.5 The Waiting Area:

The waiting area is a common resource shared between the producer and consumers. Customers can only be added as long as there is room in the waiting

area. The capacity of the waiting area is decided by *waitingAreaCapacity*.

The waiting area should have:

- A way of keeping track of all the waiting customers
- A max capacity.

Functionality:

- Let customers enter when there is room
- A way for customers to be fetched by waitresses

## 1.6 Waitress

The waitresses will work as the consumers.

When a customer leaves the waiting area. After a customer is fetched, the waitress uses some time before taking the customer's order.

After the customer is finished ordering and eating, the waitress can fetch a new customer from the waiting area.

Note that the customers should be fetched from the waiting area in a **first come – first serve** manner. This means that the earlier the customer enters the waiting area, the sooner he/she is fetched.

## 1.7 The Clock:

The clock in the bar serves two purposes:

1. It will alert the door of the closing time, so the door will generate no more customers.
2. The clock keeps track of the current time of simulation, which will be used for logging.

You will not have to implement the clock yourself; instead you must use the provided class: *Clock.java*. This clock sets the *isOpen* variable to false when the simulation time ends. The only thing you need to do is to initiate the clock (*new Clock(duration)*), and use the *isOpen* variable properly.

## 1.8 Logging:

To see what is happening in the Sushi Bar you should print out the following messages at the time they happen:

- Customer #ID is now waiting.

- Customer #ID is now fetched.
- Customer #ID is now eating.
- Customer #ID is now leaving.
- ***** NO MORE CUSTOMERS - THE SHOP IS CLOSED NOW. *****

In order to print these messages you are to use the provided *write* method in the *SushiBar.java* class. A call to the method may look something like this:

$SushiBar.write(Thread.currentThread().getName() + \text{“} : Customer'' + customerNo + \text{“}isnowcreated.'');$
Please stick to this format and only print out statistics and the messages listed above.

## 1.9 Statistics:

At the end of the simulation, you should print out some statistics.

The statistics should contain:

1. Total number of orders.
2. Total number of takeaway orders
3. Total number of orders that customers have eaten at the bar

## 1.10 Note about orders:

As mentioned, each customer has a choice to eat some orders in the bar and have the remaining as takeaway.

$NumberOfOrders = EatenOrders + TakeawayOrders$

## 1.11 Requirements

Deliverables:

1. Your complete implementation of the Sushi bar (Source Code).
2. A document explaining your implementation. In addition, you have to address the following points:
   (a) What are the functionality of *wait()*, *notify()* and *notifyAll()* and what is the difference between *notify()* and *notifyAll()*?
   (b) which variables are shared variables and what us your solution to manage them?

(c) Which method or thread will report the final statistics and how will it recognize the proper time for writing these statistics?

- You should not kill a thread. Every thread should terminate normally, when it is finished with all of its tasks. So do not use any *Stop* or *Exit* methods.
- Try manipulating the settings of the sushi bar to make sure that your program works in all situations. These settings can be the number of seats, simulation time, frequency of generating customers and length of eating time.
- Correct statistics does not necessarily mean that your implementation is correct. You have to check all outputs of your program to make sure that everything works in your simulation.