README.md

# TDT4186 Operating System

## Practical Assignment - Implementation of Sushibar with Threading

## Overview of program:

#### Sushi bar

`Sushibar.java` is the main controller and environment for the simulation. Some global important variables are set and it creates the objects `Clock` , `WaitingArea` , `Waitress` 's, `Door` and returns the final statistics for the simulation. It also starts and stops the `Door` -threads and the `Waitress` -threads (plural).

#### Customer

`Customer.java` is the object handled by the Sushibar as a whole. The goal beeing to let the order, eat, and leave.

#### Door

`Door.java` is a Thread and "Producer" in this environment. It has the task to create `Costumer` s in a uniform rate and wait if the `WaitingArea` is full.

#### Waitress

`Waitress.java` is a thread and a "Consumer" in this environment. It has the task to fetch costumers from the `WaitingArea` , take orders and provide service to the customer while they eat.

#### WaitingArea

`WaitingArea.java` is a shared resource between the "producer"( `Door` ) and "Consumer"( `Waitress` 's). It functions as a queue for the customers before the waitresses have time to fetch them. It makes the `Door` `wait()` if the `WaitingArea` is full and `notify()` the `Door` when the waitresses fetches a customer(Then we now there MUST be place for another consumer). The waitresses are told to `wait()` if they check the WaitingArea and it is empty. The `WaitingArea` uses `notifyAll()` to awake all the waitresses that are waiting(kinda like a doorbell that all can hear. It rings even if you are not waiting but busy as well).

Even though the WaitingArea is not a thread it is the dictator of access to a shared resource. This shared resource is a place where the threads have too coordinate to not mess up. It is the code in `WaitingArea.java` that dictates when the threads should `wait()` and get `notify()` .

#### Clock

`Clock.java` is a class that helps time the operation from the beginning, when it gets to zero it closes the sushibar. Which in turn make the threads start their last tasks.

#### SynchronizedInteger

`SynchronizedInteger.java` keep counters stable and is used to increment statistics for end result.

### Difference between wait(), notify(), notifyAll()

`wait()` makes the current thread to wait indefinitely(until waken up with notify())

`notify()` wakes up the wakes up a thread that are waiting with wait(), if any are.

`notifyAll()` wakes up all threads are waiting with wait(), if any are.

### Shared variables

for variables to be shared across threads. They either have to be static variables like `waitingAreaCapacity`, `doorWait`, `isOpen` etc. If one wants to change variables they have to syncronized to prevent miscommunication about updates like the `SyncronizedInteger` class provides. The `ArrayList<Customer> queue` is a shared resource which we have syncronized with the threads.

### Reporting final statistics

The `main` in `Sushibar.java` report the final statistics. We have to wait for the Sushibar to close, and then we need the statistics from the SyncronizedIntegers, but the Customers can still be waiting, ordering or eating. So to be sure the Costumers are done eating and have left the compound so we get correct statistics we make sure to wait on the `waitress`'s threads by running `waitress.join()` on all of them before printing statistics. As the door thread always terminate before the waitresses we dont have to wait for it.