SushiBar øving TDT4186 OS

a. What are the functionality of $wait()$, $notify()$ and $notifyAll()$ and what is the difference between $notify()$ and $notifyAll()$?

Svar:

Wait() har som hensikt å pause utførelsen av tråden den blir kalt fra. Ved f.eks. å kalle waitingAirea.wait() på door-tråden pauser vi door-tråden helt til waitingArea-tråden gir et signal til at door-tråden kan fortsette utførelsen. For at wait() skal fungere i koden vår bruker vi monitor-funksjonen som finnes i java.

```
synchronized(waitingArea) {
    waitingArea.wait();
}
```

Notify() og NotifyAll() har som hensikt å signalisere til en eller flere ventende tråder at de nå kan fortsette utførelsen av tråden fra der de slapp når de tidligere satt seg i en ventende «state».

Forskjellen mellom Notify() og NotifyAll() er at Notify() sender ut et «klar-signal» til *én* ventende tråd, mens NotifyAll() sender ut dette signalet til *alle* ventende tråder.

b. Which variables are shared variables and what is your solution to manage them?

Svar:

WaitingArea.queue, servedOrders, takeawayOrders, totalOrders & customerID er variables som sees på som felles ressurser i SushiBar. De 4 sistnevnte er av typen syncronizedInteger, som betyr at gjensidig utelukkelse allerede er tatt hånd om av den klassen. Her trengte jeg da altså ikke gjøre noe.

Queue fra waitingArea er en felles ressurs som aksesseres både av producer tråden for Door-objektet og alle consumer trådene til de forskjellige Waitress-objektene. Måten jeg løste dette på var å definere metodene for å interagere med denne felles ressursen som «synchronized». Dette gjorde funksjonaliteten på queue til atomiske funksjoner, som at flere kunne hente elementer eller skrive til queue samtidig.

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

Main-tråden (hovedtråden) i SushiBar objektet er tråden som skriver ut den endelige statistikken. For å være sikker på at alle trådene var ferdig å kjøre join-et jeg hovedtråden med alle de andre trådene jeg hadde definert. Ved å joine disse etter tur venter hovedtråden til hver enkelt join-tråd er

helt ferdig å kjøre for den går videre til neste. Først etter dette har kjørt printer vi ut statistikken fra hovedtråden, og vi kan da være sikker på at alle andre tråder er ferdigkjørt.

gjennomgang, klasse for klasse, metode for metode, hvor du forklarer hva programmet gjør og hvorfor du har valgt å gjøre det på denne måten. Bruk spesifikke eksempler for å forklare håndtering av eventuelle spesielle tilfeller.

Gjennomgang av kode:

Customer:

Customer representerer kundene som går inn døra, venter i waitingArea og bestiller mat. Kundene spiser i sushi baren i en randomisert tidsperiode, før de går og det blir plass til en ny kunde.

```
public void order() {
    // nextInt range is inclusive 0 and exclusive max, so we must add "1"
    // randomly generates a number which represents the total amount of orders a
customer makes
    int totalOrders = new Random().nextInt(SushiBar.maxOrder) + 1;

    // randomly generates a number which represents how many orders of the total
amount that are served "in house"
    int inHouseOrders = new Random().nextInt(totalOrders+1);

    // Send amount of orders to update statistics for the sushi bar
    SushiBar.totalOrders.add(totalOrders);
    SushiBar.servedOrders.add(inHouseOrders);
    SushiBar.takeawayOrders.add(totalOrders-inHouseOrders);

    SushiBar.write(Thread.currentThread().getName()+" Customer:
"+this.getCustomerID()+" is now eating");

    // Represents the time the customer is sitting in the sushi bar, eating.
    int eatingTime = new Random().nextInt(SushiBar.customerWait/2) +
SushiBar.customerWait/2 + 1;
    try {
        Thread.sleep(eatingTime);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    SushiBar.write(Thread.currentThread().getName()+" Customer:
"+this.getCustomerID()+" is now leaving");
}
```

order() står for all funksjonalitet som det er verdt å nevne i klassen. Kunde bestiller et randomisert forhold mellom in-house ordre, som spises innendørs, og take-away. Jeg har valgt å legge intervallet det tar en kunde å spise på halve- til hele tiden som er spesifisert i customerWait.

Door:

Door er produceren i koden og generer kunder som umiddelbart «går inn døra». Er venteområdet fullt blir door-tråden satt i en ventende status til venteområdet signaliserer at det er plass.

```
@Override
public void run() {
    while(SushiBar.isOpen){
        if(waitingArea.isFull()){
            try {
                // wait until the thread of waitingArea gives a notification
                synchronized(waitingArea) {
                    waitingArea.wait();
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        else{
            // Randomly generate the time it takes for the door to "generate" a
new customer
            // generated using the "doorWait" variable in the SushiBar class
            int randomWait = new Random().nextInt(SushiBar.doorWait);
            try {
                Thread.sleep(randomWait);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            // Creates a new customer after "randomWait" time has passed
            Customer customer = new Customer();
            SushiBar.write(Thread.currentThread().getName()+
                    " Customer: "+customer.getCustomerID()+" is now waiting");
            this.waitingArea.enter(customer);
        }
    }
    SushiBar.write("***** DOOR CLOSED. *****");
}
```

Run() kjøres når vi kaller start() på tråden som inneholder runnable klassen Door.

Run() står for all funksjonalitet som er verdt å nevne.

I koden skiller jeg mellom to tilfeller:

1. venteområdet er fullt
2. venteområdet er ikke fullt

for 1. settes døren på vent

for 2. går det en randomisert mengde tid før en kunde produseres og sendes inn i venteområdet.

Waitress

Waitress trådene representerer consumer-en og tar kunder ut av venteområdet så «fort de kan». Hvor fort dette blir, er randomisert ved bruk av waitressWait

```
public void run() {
    // If the sushi bar is still open or we have remaining customers after the
door has closed
    // we still have orders to take
    while(SushiBar.isOpen || !waitingArea.isEmpty()){
            if(waitingArea.isEmpty()){
                try {
                    // Wait until the thread of waitingArea gives a notification
                    synchronized(waitingArea) {
                        waitingArea.wait();
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            } else{
                // Represents the customer that is first in line in queue in the
waiting area
                Customer customer = waitingArea.next();
                SushiBar.write(Thread.currentThread().getName()+
                        " Customer: "+customer.getCustomerID()+" is now fetched");

                // Randomly generate the time it takes for a customer to receive
an order they have made
                // generated using the "waitressWait" variable in the SushiBar
class
                int orderingTime = new Random().nextInt(
                        SushiBar.waitressWait/2) + SushiBar.waitressWait/2 + 1;
                try{
                    Thread.sleep(orderingTime);
                }catch (InterruptedException e) {
                    e.printStackTrace();
                }
                customer.order();

            }
    }
```

vi kjører løkka så lenge ikke vi har kunder igjen eller sushi baren er åpen.

Vi skiller mellom to tilfeller:

1. venteområdet er tomt
2. venteområdet er ikke tomt

for 1. settes den aktuelle waitress-tråden på vent til den får signal fra waitingArea-tråden.

For 2. plukkes en customer ut av waitingArea ved å kalle waitingArea-next(). Vi bruker så en randomisert tid til å ta bestilling og levere denne bestillingen.

WaitingArea

WaitingArea er klassen som sees på som bufferet mellom producer og consumer. Den sier ifra til consumer om det er tomt i bufferet og til producer at det er fullt i bufferet

enter

```
public synchronized void enter(Customer customer) {
    // Adds the customer to the queue that represents the waiting area
    this.queue.add(customer);
    // Notifies all threads in this threads monitor. In real time only waitress-
threads should be waiting
    // for a customer to "walk in the door"
    notifyAll();
}
```

Det vil være door som gjennom assosiasjon kaller enter på dens tilhørende waitingArea. Hvis enter er kallet kan det ikke være fullt i waitingArea-queuen og derfor kan ikke door-tråden være en av de ventende trådene som blir alarmert via notifyAll(). Når en kunde «ankommer» waitingArea vil da altså waitress-tråder blir alarmert om det skulle være noen ventende.

Next

```
public synchronized Customer next() {
    // Removes the person "first in line" at the queue in the waiting area,
freeing up a space
    Customer nextCustomer = this.queue.pop();
    // Notifies all threads in this threads monitor. In real time only the door
thread should be waiting
    // for a free space in the waiting area
    notifyAll();

    return nextCustomer;
}
```

Det vil være en waitress-tråd som gjennom assosiasjon kaller next på dens tilhørende waitingArea. Hvis next er kallet kan det ikke være tomt i waitingArea-queuen og derfor kan ikke en waitress-tråd være en av de ventende trådene som blir alarmert via notifyAll(). Dette kommer av at det er plass til 20 stk. i waitingArea og kun 8 waitresses til betjening. Når en kunde fjernes fra waitingArea (betjenes) vil da altså door-tråden bli alarmert om den skulle være noen ventende.

SushiBar

```
public static void main(String[] args) {
    log = new File(path + "log.txt");

    // Initializing shared variables for counting number of orders.
    totalOrders = new SynchronizedInteger(0);
    servedOrders = new SynchronizedInteger(0);
    takeawayOrders = new SynchronizedInteger(0);
    customerID = new SynchronizedInteger(1);

    // *** Initializing all needed objects and threads for the Sushi Bar to work
***
    new Clock(SushiBar.duration);
    WaitingArea waitingArea = new WaitingArea(waitingAreaCapacity);

    // Each waitress will have their own thread, putting them in this list for
easy, safe termination
    List<Thread> waitresses = new ArrayList<>();

    // Create a thread for every waitress. Number of waitresses indicated by
waitressCount
    for (int i=0;i<waitressCount;i++){
        // Assigns a thread for each waitress (each consumer)
        Thread waitress = new Thread(new Waitress(waitingArea));
        waitress.start();
        waitresses.add(waitress);
    }

    // The door (producer) is executed in this thread
    Thread door = new Thread(new Door(waitingArea));
    door.start();

    // Joining each individual thread with this main thread to safely ensure that
all threads are done executing.
    for (Thread waitressThread : waitresses) {
        try {
            waitressThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    try {
        door.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    write("***** NO MORE CUSTOMERS - THE SHOP IS CLOSED NOW. *****");

    SushiBar.write("Total number of orders: " + totalOrders.get());
    SushiBar.write("Orders eaten here: " + servedOrders.get());
    SushiBar.write("Orders taken out: " + takeawayOrders.get());
}
```

Dette er hovedklassen og det er verdt å nevne at main-metoden kjøres i sin egne tråd. Denne tråden blir hovedtråden og er tråden som lager trådene til Waitress-instansene og Door-instansen.

Føler at funksjonaliteten står godt forklart som kode i teksten.