1. Description of all classes

Customer.java

Variables:

- int:
  - id = The id number of each created instance of customer
  - customerCount = simple counter that works with the id, incremented with each created Customer()
  - numOrders = number of orders each customer "decides" to order
  - takeAwayOrders = number of take away orders
  - eatInOrders = number of orders eaten at the bar

Functions:

- order(): Each customer orders a random amount of order, the orders can be eaten in or taken away. The variable p is determining whether customer eats in or takes away.
  Than I add the number of orders to the synchronized integers which I store in SushiBar.java and finally return the number of orders customer orders.
- GetCustomerID(): returns the *int* id of each customer

WaitingArea.java

The data structure *Queue* is a FIFO. That is ideal to store our Customers.

- Variables:
  - int maxCapacity is the maximum number of items in the waitingArea
- Functions:
  - *Synchronized* **enter**(Customer): Makes customer able to enter the waiting area. If the waitingArea has already got to it's limit we set the customer/door thread to blocked position. If the SushiBar is open we wake up the thread and add the customer to the waitingArea.
  - *Synchronized* **next()** if the situation occurs where the bar is open but the waitingArea is empty we put the thread to blocked position. But if it's not empty we return the next customer in line.
  - *Synchronized* **isEmpty()** simple isEmpty function that returns true if nobody is in the waiting queue.

Door.java

Door implements Runnable which means it is a thread, and it works as a producer. In Door.java we have instances of waitingArea.java and Customer.java
Door is implemented with waitingArea.

Functions:

- run(): creates a thread while the SushiBar is open determined by the boolean variable isOpen, and adds the customer to the waitingArea and waits the standard doorWait.
- *Synchronized* createCustomer(): Simple function that creates a new instance, and returns, of the class Customer.java

Waitress.java
Waitress implements Runnable which means it is a thread, and it works as a consumer.
If the SushiBar is open and the waitingArea has some customers waiting we take the next customer and serve him. We wait a little bit and than the customer orders with the *order()* function from the Customer.java. We wait a little bit while the customer eats.

SushiBar.java
In the main class I created a new instance of all the classes above and start the threads. And start the clock with the duration given. We create waitress by loop over the amount of waitressCount. And finally print out all key figures and display the statistics with the write function.

2a. wait(), notify() and notifyAll()

- ✔ wait(): Makes the current thread give up the lock on the monitor so that another thread can now other thread can access the monitor and calls notify() to take the tread from Blocked state to Ready.

- ✔ notify(): Wakes up the thread that has been put to *blocked* by wait.

- ✔ NotifyAll(): Wakes up all threads that called wait() on the same object.

2b. Which variables are shared variables and what us your solution to manage them?

We have shared variables as customerCounter, servedOrders, takeawayOrders and totalOrders. What means that they are shared is that more than one thread tries to access them and update throughout the program. Our solution is synchronizing them. We use the keyword in Java synchronize, than only one thread can access the variable at a time and others are blocked from execution until the first thread is finished with the object.

2c. Which method or thread will report the final statistics and how will it recognize the proper time for writing these statistics?

4 secs(*SushiBar.duration*) that customers are created, and after that the timer has scheduled a Thread called RemindTask() which runs after that 4 secs.
The SushiBar.isOpen boolean variable is switched to false which closes the SushiBar and the Door stops producing Customers.
Finally the timer.cancel() is called. The description of cancel() is “Does not interfere with a currently executing task (if it exists). Once a timer has been terminated, its execution thread terminates gracefully, and no more tasks may be scheduled on it.”