

PROBLEM STATEMENT: FOOD OUTLET MANAGEMENT SYSTEM

Goal: The goal of this assignment is to get some practice with combining the concepts learnt so far to choose appropriate data structures for an efficient implementation of a simulation environment.

Problem Statement: You are the owner of a big and famous burger restaurant called McMohans Burgers. The restaurant has K billing counters numbered 1 to K. Since your restaurant is a hit amongst youngsters, you get a lot of customers leading to long queues in billing as well as food preparation. You would like to know some statistics like average waiting time, average queue length etc., so that proper steps to improve customer convenience can be taken. You want to do this by running a simulation, as it is cost/time efficient and allows for a range of experimentation on the assumed model.

Assume that customers arrive randomly. A new customer always joins the billing queue with the smallest length at that time. If there are multiple billing queues with the same smallest lengths, then the lowest numbered queue of those is chosen by the customer. If two customers arrive at the same time, you may assume that they came one after the other at the same time instant, i.e., the first customer will already be in some queue, when the second customer is deciding which queue to join.

Different employees servicing the different billing queues are not equally efficient. The billing specialist (who takes the order and processes payment for a customer) in billing queue k will take k units of time in completing the order. After the order is completed, the customer order is printed automatically and sent to the chef, who prepares the burgers in the sequence he/she receives the orders. If two orders arrive simultaneously then the chef chooses the order from the higher numbered billing queue first.

The chef has a large griddle on which at most M burger patties can be cooked simultaneously. Each burger patty gets cooked in exactly 10 units of time. Whenever a patty is cooked another patty starts cooking in the same time instant. Example, if a patty starts to be cooked at time 10, then it completes at time 20 and another patty starts cooking also at time 20.

Upon cooking, the burger is delivered to the customer in one unit of time. Whenever a customer gets all their burgers, they leave the restaurant instantaneously (it's a take-away restaurant with no dine-in).

Your goal is to simulate this whole process. The simulation has to be driven by events. Events in our simulation environment are arrival/departure of a customer, completion of payment for an order, completion of one or more burgers, etc. For each customer, you have to maintain their state: waiting in queue, waiting for food, or left the building. You will also have to maintain a global clock, which will move forward after all events of a given time point are simulated. You should assume time as discrete integers starting at 0.

If there are multiple events happening at the same time instant, the events are executed in the following order:

1. Billing specialist prints an order and sends it to the chef; customer leaves the queue.
2. A cooked patty is removed from the griddle.

3. The chef puts another patty on the griddle.
4. A newly arrived customer joins a queue.
5. Cooked burgers are delivered to customers.

If there is a query at the same time instant, the query must be answered after all the events for the time instant have been executed in the above-mentioned order.

Your goal of the assignment is to implement this simulation environment through the following operations:

```
public interface MMBurgersInterface {
```

```
    public boolean isEmpty();           /* Returns true if there is no further event to
    simulate */
```

```
    public void setK(int k) throws IllegalArgumentException; /* The number of
    billing queues in the restaurant is k. This will remain constant for the
    whole of simulation */
```

```
    public void setM(int m) throws IllegalArgumentException; /* At most m burgers
    can be cooked in the griddle at a given time. This will remain constant for
    the whole of simulation */
```

```
    public void advanceTime(int t) throws IllegalArgumentException; /* Run the
    simulation forward simulating all events until (and including) time t. */
```

```
    public void arriveCustomer(int id, int t, int numb) throws
    IllegalArgumentException; /* A customer with ID=id arrives at time t. They want
    to order numb number of burgers. Note that an id cannot be repeated in a
    simulation. Time cannot be lower than the time mentioned in the previous
    command. Numb must be positive. ID can be any value and does not have to be
    consecutive integers. */
```

```
    public int customerState(int id, int t) throws IllegalArgumentException; /*
    Print the state of the customer id at time t. Output 0 if customer has not
    arrived until time t. Output the queue number k (between 1 to K) if customer
    is waiting in the kth billing queue. Output K+1 if customer is waiting for
    food. Output K+2 if customer has received their order by time t. Note that
    time cannot be lower than the time mentioned in the previous command. All
    commands till time t have been already sent to the environment */
```

```
    public int griddleState(int t) throws IllegalArgumentException; /* Print the
    number of burger patties on the griddle at time t. Note that t cannot be
    lower than the time mentioned in the previous command. All commands till time
    t have been already sent to the environment */
```

```
    public int griddleWait(int t) throws IllegalArgumentException; /* Print the
    number of burger patties waiting to be cooked at time t. I.e., number of
    burgers for which order has been placed but cooking hasn't started. Note that
```

*t cannot be lower than the time mentioned in the previous command. All commands till time t have been already sent to the environment */*

```
public int customerWaitTime(int id) throws IllegalArgumentException; /* Print the total wait time of customer id from arriving at the restaurant to getting the food. These queries will be made at the end of the simulation */
```

```
public float avgWaitTime(); /* Returns the average wait time per customer after the simulation completes. This query will be made at the end of the simulation. */
```

```
}
```

Write a program which implements such a data-structure. High credit will be given to choice of proper data-structures and efficiency. As a rule of thumb, whenever you need to store data which will be accessed in future via some integer key, use balanced search trees. Whenever you need to repeatedly find the smallest key in a set, use heaps. Whenever you need to implement a last in first out sequence use stacks. Whenever you need to implement a first in first out sequence use queues. Whenever you have a set of data points on which you need to iterate, you may use an array or a list. (Hint: in this assignment, you need at least one heap).

Example 1 -

Let Number of counters (K) = 3

Let Size of griddle (M) = 6

Initially, all the three counters are empty.

At t=0,

Customer 1 comes with an order of 3 burgers.

Customer 2 comes with an order of 4 burgers.

Customer 3 comes with an order of 5 burgers.

According to the condition of allocating customers to counters.

Customer1 goes to counter1 (q1),

Customer2 goes to counter2 (q2)

Customer3 goes to counter3 (q3)

Billing specialist in counter 1 takes 1 unit of time for placing an order.

Billing specialist in counter 2 takes 2 units of time for placing an order.

Billing specialist in counter 3 takes 3 units of time for placing an order.

1. Customer1 waits in counter1 for 1 unit of time (from $t = 0$ to 1).
2. At $t = 1$, 3 burgers of customer1 are put on the griddle .
3. At $t = 2$, order of Customer2 is printed and sent to the chef. 3 out of 4 burgers of Customer2 are put on griddle.
4. At $t = 3$, order of Customer3 is printed and sent to the chef.
5. At $t = 11$, 3 burgers of Customer1 get cooked. Now, 3 spaces are free. 1 burger of Customer2 and 2 burgers of Customer3 are put on the griddle.
6. At $t = 12$, 3 burgers of Customer2 are cooked. Now, 3 spaces are free. 3 burgers of Customer3 are put on the griddle. 3 burgers of Customer1 are delivered and Customer1 leaves.
7. At $t = 13$, Customer2 receives 3 burgers.
8. At $t = 21$, 1 burger of Customer2 and 2 burgers of Customer3 get cooked.
9. At $t = 22$, Customer3 receives 2 burger and its remaining 3 burger get cooked. Order of Customer2 is delivered and Customer2 leaves.
10. At $t = 23$, Customer3 receives all burgers and leaves.

Finish time of Customer1 = 12 units

Finish time of Customer2 = 22 units

Finish time of Customer3 = 23 units

Input Output for example 1

[In] : arriveCustomer(1, 0, 3)

[In]: arriveCustomer(2, 0, 4)

[In]: arriveCustomer(3, 0, 5)

[In] customerState(2 , 1)

[Out]: 2

```
[In]: griddleState(1)

[Out]: 3

[In]:griddleWait(1)

[Out]: 0

[In]: griddleState(2)

[Out]: 6

[In]: customerState( 1, 3)

[Out]: 4

[In] customerState(2, 7)

[Out]: 4

[In]:griddleWait(10)

[Out]: 6

[In]: griddleState(14)

[Out]: 6

[In]: griddleState(21)

[Out]: 3

[In]: isEmpty()

[Out]: False

[In]: advanceTime(23)

[In]: isEmpty()

[Out]: True

[In]: customerWaitTime(1)

[Out]: 12

[In]: customerWaitTime(2)

[Out]: 22

[In]: customerWaitTime(3)
```

[Out]: 23

[In]: avgWaitTime()

[Out]: 19.00

Example 2:

Let no. Of counters (K) = 2

Let size of griddle (M) = 8

Initially, all the counters are empty.

1. Customer1 comes at t = 0 with order of 5 burgers.

2. Customer2 comes at t = 0 with order of 6 burgers.

3. Customer3 comes at t = 1 with order of 4 burgers.

4. Customer4 comes at t = 1 with order of 5 burgers.

Billing specialist in counter 1 take 1 unit of time for placing an order.

Billing specialist in counter 2 take 2 units of time for placing an order.

1. At t = 0, Customer1 and Customer2 arrive. Customer1 goes to counter1 and Customer2 goes to counter2.

2. At t = 1, Order of Customer1 is placed and sent to the chef. 5 burgers are put on the griddle. Customer3 and Customer4 arrive. Customer3 goes to counter1. As counter1 and counter2 have the same length, Customer4 goes to counter1.

3. At t = 2, Order of Customer2 and Customer3 are placed and sent to the chef. Chef considers order of customer2 (high counter value) first and put 3 burgers of customer2 on the griddle.

4. At t = 3, Order of Customer4 is placed and sent to the chef.

Chef's Queue - (2,3) -> (3,4) -> (4,5)

where (val1, val2) = (customer id, number of burgers not on griddle)

5. At t = 11, 5 burgers of Customer1 are cooked are removed from griddle. Now 3 burgers of customer2 and 2 burgers of customer3 are put on griddle.

Chef's Queue - (3,2) -> (4,5)

6. At $t = 12$, 3 burgers of customer2 are cooked. Now, 2 burgers of customer3 and 1 burger of customer4 are put on griddle. Order of customer1 is delivered and he leaves.

Chef's Queue - (4,4)

7. At $t = 13$, 3 burgers of customer2 are delivered to him.

8. At $t = 21$, 3 burgers of customer2 and 2 burgers of customer3 are cooked. Now, 4 burgers of customer4 are put on griddle.

Chef's Queue - empty

9. At $t = 22$, 2 burgers of customer3 and 1 burger of customer4 are cooked. Order of customer2 is finished and he leaves.

10. At $t = 23$, Order of customer3 is finished and he leaves.

11. At $t = 31$, 4 burgers of customer4 are cooked.

12. At $t = 32$, Order of customer4 is delivered and he leaves.

Finishing time of customer1 = 12 units

Finishing time of customer2 = 22 units

Finishing time of customer3 = 23 units

Finishing time of customer4 = 32 units

Input Output for example 2

[In]: arriveCustomer(1, 0, 5)

[In]: arriveCustomer(2, 0, 6)

[In]: arriveCustomer(3, 1, 4)

[In]: arriveCustomer(4, 1, 5)

[In]: customerState(3, 1)

[Out]: 1

[In]: customerState(4, 1)

[Out]: 1

[In]: griddleState(1)

```
[Out]: 5

[In]: customerState(1, 5)

[Out]: 3

[In]: griddleState(5)

[Out]: 8

[In]: customerState(2, 6)

[Out]: 3

[In]:griddleWait(6)

[Out]: 12

[In]:griddleWait(12)

[Out]: 7

[In]: griddleState(25)

[Out]: 4

[In]: advanceTime(32)

[In]: isEmpty()

[Out]: True

[In]: customerWaitTime(1)

[Out]: 12

[In]: customerWaitTime(2)

[Out]: 22

[In]: customerWaitTime(3)

[Out]: 22

[In]: customerWaitTime(4)

[Out]: 31

[In]: avgWaitTime()

[Out]: 21.75
```


Note that while in both examples, customers arrived first, and all queries regarding state etc were issued later, this will not be in true in general. And customer arrivals and queries will be interleaved.

What is being provided?

Your assignment folder contains 5 java files:

1. `IllegalNumberException.java` : Defines a custom exception (Do not modify this file)
2. `MMBurgersInterface.java` : Defines the simulation interface (Do not modify this file)
3. `MMBurgers.java` : You have to implement all the functions of `MMBurgersInterface` in this file and you can add your own classes and methods also.
4. `Tester1.java` : File with the main function to test example1.
5. `Tester2.java` : File to test example2.

How to run?

The starter folder contains a Makefile with the necessary commands to run the program, if you are unable to run the makefile just copy-paste the commands in the `all:` section and run sequentially in the shell.