NUS WebMail IVLE LIBRARY MAPS

Search search for... in NUS Websites GO

CodeCrunch

Home | My Courses | Browse Tutorials | Browse Tasks | Search | My Submissions | Logout | Logged in as: **e0959123**

CS2030 (2310) Lab #1

Tags & Categories

Tags:

Categories:

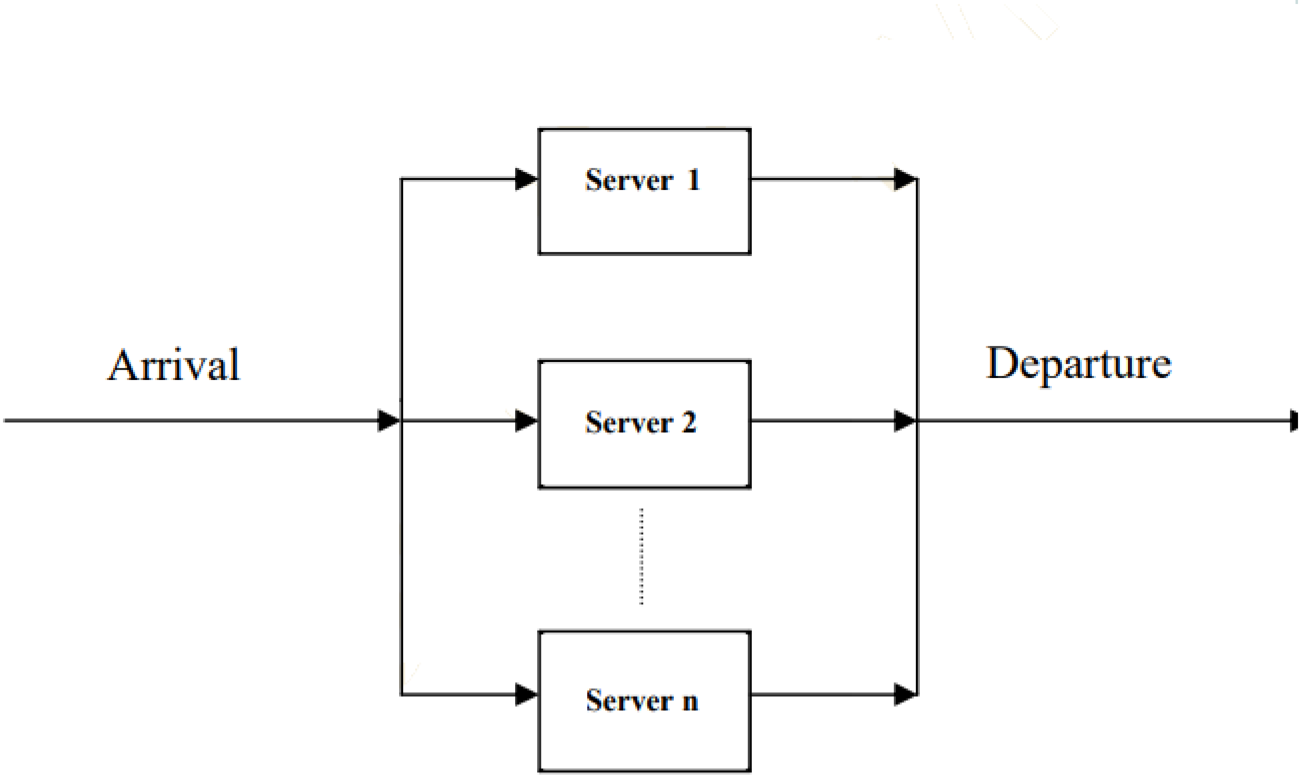
Related Tutorials

Task Content

Lab #1

We would like to simulate how customers are being served by servers. Specifically, a customer that arrives will look for the first available server, and he/she will be served for some time. On seeing that all servers are busy, a new arriving customer will just leave.

The following depicts a schematic of the simulation:



- There are *n* servers.
- Each server can serve one customer at a time.
- Each customer has a service time (time taken to serve the customer).
- When a customer arrives:
  - the servers are scanned from 1 to n to find the first one that is available (not serving any customer) and gets served immediately.
  - if all servers are serving customers, then the customer just leaves.

As an example, given an input of two servers followed by the arrival and service times of three customers,

```
2
0.500 1.000
0.600 1.000
0.700 1.000
```

a simulation can be performed with the following output that shows how each customer is served.

```
0.500 customer 1 arrives
0.500 customer 1 served by server 1
0.600 customer 2 arrives
0.600 customer 2 served by server 2
0.700 customer 3 arrives
0.700 customer 3 leaves
[2 1]
```

The last line of output shows the number of customers served and left respectively.

A working program is provided for you comprising of

- a non-OO implementation `Main.java`, and
- the utility class `ImList.java`.

Note that the following alternative `Main` class will be used to test your objected-oriented implementation.

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ImList<Double> arrivalTimes = new ImList<Double>();
        ImList<Double> serviceTimes = new ImList<Double>();

        int numOfServers = sc.nextInt();
        while (sc.hasNextDouble()) {
            arrivalTimes = arrivalTimes.add(sc.nextDouble());
            serviceTimes = serviceTimes.add(sc.nextDouble());
        }

        Simulator sim = new Simulator(numOfServers, arrivalTimes, serviceTimes);
        System.out.println(sim.simulate());
        sc.close();
    }
}
```

Sample runs of the program is given below. You should create your own input files using `vim`.

```
$ cat 1.in
3
0.500 1.000
0.600 1.000
0.700 1.000

$ java Main < 1.in
0.500 customer 1 arrives
0.500 customer 1 served by server 1
0.600 customer 2 arrives
0.600 customer 2 served by server 2
0.700 customer 3 arrives
0.700 customer 3 served by server 3
[3 0]

$ cat 2.in
3
0.500 1.000
0.600 1.000
0.700 1.000
1.500 1.000
1.600 2.000
1.700 3.000

$ java Main < 2.in
0.500 customer 1 arrives
0.500 customer 1 served by server 1
```

```
0.600 customer 2 arrives
0.600 customer 2 served by server 2
0.700 customer 3 arrives
0.700 customer 3 served by server 3
1.500 customer 4 arrives
1.500 customer 4 served by server 1
1.600 customer 5 arrives
1.600 customer 5 served by server 2
1.700 customer 6 arrives
1.700 customer 6 served by server 3
[6 0]

$ cat 3.in
2
0.500 1.000
0.600 1.000
0.700 1.000

$ java Main < 3.in
0.500 customer 1 arrives
0.500 customer 1 served by server 1
0.600 customer 2 arrives
0.600 customer 2 served by server 2
0.700 customer 3 arrives
0.700 customer 3 leaves
[2 1]

$ cat 4.in
2
0.500 1.000
0.600 1.000
1.500 1.000

$ java Main < 4.in
0.500 customer 1 arrives
0.500 customer 1 served by server 1
0.600 customer 2 arrives
0.600 customer 2 served by server 2
1.500 customer 3 arrives
1.500 customer 3 served by server 1
[3 0]

$ cat 5.in
2
0.500 1.100
0.600 0.900
0.700 0.700
1.500 0.100
1.600 0.200
1.700 0.300

$ java Main < 5.in
0.500 customer 1 arrives
0.500 customer 1 served by server 1
0.600 customer 2 arrives
0.600 customer 2 served by server 2
0.700 customer 3 arrives
0.700 customer 3 leaves
1.500 customer 4 arrives
1.500 customer 4 served by server 2
1.600 customer 5 arrives
1.600 customer 5 served by server 1
1.700 customer 6 arrives
1.700 customer 6 served by server 2
[5 1]
```

Here is how you should go about developing your program:

- Using pen and paper, design the `Simulator` and all other classes that will be used in the solution. Model the relationships (e.g. has-a) between classes.
- Ensure that these classes are not cyclic dependent so that bottom-up testing can proceed:
  - Start by writing the simplest class and test it using JShell. Make sure it works according to your specifications before proceeding.

    - Following incremental development and testing, write the next class that depends on the working class(es). Write tests to incrementally test this new class and its dependence on other classes defined earlier.
- Keep to the spirit of effect-free programming when designing your solution:
    - All objects cannot change state after it is created. State changes are considered as *side effects*. Any change in state should be returned as a new object.
    - Input and output are also considered side effects. Rather than allowing input and ouput throughout the program run, read all inputs at the beginning of the program, and output only at the end of the program. This has been done for you in the `Main` class.

Submission (Course)

Select course: CS2030 (2023/2024 Sem 1) - Programming Methodology II

Your Files:

BROWSE

SUBMIT (only .java, .c, .cpp, .h, .jsh, and .py extensions allowed)

To submit multiple files, click on the Browse button, then select one or more files. The selected file(s) will be added to the upload queue. You can repeat this step to add more files. Check that you have all the files

Terms of Use | Privacy | Non-discrimination

MySoC | Computing Facilities | Search | Campus Map

School of Computing, National University of Singapore