Bank Queue Simulation

Queue simulation in the service counters of a bank using Java

S M Asiful Islam Saky

Report

Contents

1. Introduction

1.1 Purpose of the Project

This project is aimed at designing a Java program that emulates the queue management in a bank as a means of designing a system that meets the core objectives of efficient customer service and tackling long queues. Therefore, based on the palette of known approaches to queue management, the goal is to model what actually happens in bank branches and determine how the identified factors affect the efficiency of the work in general. This simulation will be very useful as it will offer a setup for displaying the behavior of queues in banks that will eventually assist in the development of methods for creating better experiences and efficient procedures in the bank.

1.2 Background Information

It is important to emphasize that approaches to managing queues are important and significant for the functioning of banks. Long waiting times are considered to be very determinantal to consumers: hence, consumers are likely to desert a bank when they experience delayed services. On the other hand, the effective management of the queue appeals to the customers and eases service delivery processes. In this way they start with different rates and different numbers of service coun ters, ascertain the impact of the service time and other parameters. This type of analysis enables the identification of the limitations and the actual experimentation of the solutions on the organisation without compromising real-time running of the business.

The topic of queue management deals with the way and with which the population is served and dened in particular points. In banking this means cutting down on the time that a client spends waiting for a banker, fair and efficient distribution of a banker’s time and effort, and the optimal use of available resources. It is therefore advisable for the service providers to ensure that the queue management is well designed bearing in mind the high Returns on Investment that are associated with well-coordinated service provision, improved client satisfaction and efficient operations.

1.3 Objectives

- **Develop a Java-Based Simulation:** Develop a good QR which mimics the process of organizing a line of customers in a given bank using Java.

- **Performance Analysis:** Review how the queue management system has fared in the various scenarios to discover which mattered most when it comes to the rate of flow and customer satisfaction.
- **Optimization Insights:** Suggest possible futures for the simulation experiment relating to queuing management in an actual banking context and make specific recommendations as to how the queue may be optimized.

In pursuit of these goals, the involvement of enrolled clients is expected to foster increased efficiency in queue management within the banking project with a view of steering up the quality of service delivery excellence and organizational performance.

2. Project Requirements

2.1 Functional Requirements

The functional requirements can be summarized as the simulation of a bank's service counters or identifying the key measures that define the dynamics of a bank's service counters. The objectives of the simulation include: The simulation seeks to emulate the pattern of unit arrival, service time, and queue handling procedures.

2.1.1 Input Parameters

The simulation requires specific input parameters to initialize and run accurately:The simulation requires specific input parameters to initialize and run accurately:

i. Number of Service Counters: This parameter defines the number of counters that are effective and ready to serve banking customers at any given time. Every service counter is unique, and it is possible to include that each counter can have its own line or separate line of customers.

ii. Number of Customers: This input determines the total number of customers that are expected to visit the particular bank and conduct their respective business in the course of the simulation. It will simulate each customer throughout their purchasing journey up to the time they receive their services.

iii. Time Each Customer Will Take at the Counter: This parameter describes the time which it takes to attend to a customer in counter or service counter time for each customer. It mainly refers to the number of minutes that a teller takes to attend to a customer which also includes completing the transactions of the particular customer besides responding to any questions or attending to the other needs of the customer with regards to the usage of the services of the specific teller.

iv. Rate of Customer Arrival at the Bank: This parameter gives the proportion of the time that is taken between each arriving customer to another at the bank. Because it is usually based on one or more customers at a time, it is often measured in terms of how long it takes for the next customer to arrive, for instance, two minutes per customer.

2.1.2 Simulation Process

The core of the simulation involves managing customer queues and service processes efficiently:The core of the simulation involves managing customer queues and service processes efficiently:

i. Customer Assignment to Service Counters: In the case of many customers' arrival, the newly arrived customers are automatically brought to the service counter which has comparatively few clients. As stated in the previous method, this one seeks to distribute workloads evenly across counters, and reduce time that customers spend waiting.

ii. Simulation Termination: The simulation produces a number of important performance indices that in turns offer important informations regarding the conduct and performance of the bank's service counters:

1. Time Taken to serve all the Customers = Total time taken to complete all the Service + any other time spent to accommodate Customers.
2. On number of customers served by each service counter:
3. Average service time, customer, service counter
4. First, let us turn our focus to the average time customers were made to wait before being served

2.2 Non-functional Requirements

These include usability and user interface, which are a critical element of the non-functional requirements as they provide a basis for a reliable and easily navigable simulation.

User Interaction: All user inputs must be interactive.

Performance: The program should efficiently manage and process the queues.

Documentation: The code should be well-documented with comments along testing, and results.

3. System Design

The design of the bank queue simulation is structured around several key components: In this program there are the Queue data structure, the Customer class, the ServiceCounter class, and the BankSimulation class. All these components collectively provide a life-like experience of a customer service scenario within the institutional context of a bank. That is why the system is using the features of queues to coordinate the customer flow, providing the customer service to every client as soon as possible.

3.1 Queue Data Structure

In the middle of the simulation, there is the Queue data structure which plays the key role in determining customers serving process sequence. In this particular project, the Queue interface implemented in Java is leveraged, which is based on the LinkedList class. Since the Queue interface maintains an array-like structure where they allow the first element to be removed and the last element to be added, this makes a style of serving customers in a more realistic and real life situation. This is because the LinkedList implementation also guarantees the efficiency of the insertion and deletion operations, which are important functions in the queue operations where customers continue to join the queue and are being served.

3.2 Customer Class

The Customer class represented a single consumer of products and services in the simulation. Customers are depicted on an activity axis by attributes like customer arrival time, service time and time customer starts to be serviced. The service time is written as the total time the customer is expected to spend outside the service counter, while the arrival time is the time when the customer sets foot in the bank. This is the time from when the customer is initiated with the service to be taken till the service is completed. This class captures the customer and their activities in the system, thereby simplifying the task of policing each of the customers within the system.

3.3 ServiceCounter Class

ServiceCounter defines each counter of the services within the bank especially when determining the charges to be paid for the services. Every counter keeps own lists of the clients with two metrics: the sum of service time and amount of customers. It includes methods to add the customer to the queue and process the queue as many customer serving elements are present. After all services have been provided to a certain customer, that customer is expelled from the queue, and the total time taken as well as the number of customers are amplified. This design guarantees that each service counter works as a distinct model, which matches the nature of distinct bank counters as performed in real life.

3.4 BankSimulation Class

The feature that is most important in the simulation is controlled by the BankSimulation class. It creates and sets the initial conditions of the simulation, including the number of service counters, the number of customers, the time it takes any one specific customer to be served, and the rate at which customers arrive at the service-counter. Customers also arrive at the class and the class responsible for placing them in the queue appropriately depending on the time intervals. At the end of each of the simulation steps, the class examines each of the service counters and proceeds to process the entities in the respective queues before moving ahead in the simulation clock until all the customers are out of the queuing system.

4. System Implementation

4.1 Queue Implementation

LinkedList class in Java is employed to develop the queue for every service counter because the dynamic data structure operates efficiently. There are two queue operations; which is putting a customer to the end of the line and removing him or her from the front of the line and onto the service line after completing the service. This way it assists in ensuring that customers are served in the order of their arrival thus creating a FIFO base.

```java
import java.util.LinkedList;
import java.util.Queue;

public class ServiceCounter {
    private Queue<Customer> queue;
    private int totalServiceTime;
    private int customersServed;

    public ServiceCounter() {
        queue = new LinkedList<>();
        totalServiceTime = 0;
        customersServed = 0;
    }

    public void addCustomer(Customer customer) {
        queue.add(customer);
    }

    public void processQueue(int currentTime) {
        if (!queue.isEmpty()) {
            Customer customer = queue.peek();
            if (customer.getServiceStartTime() == 0) {
                customer.setServiceStartTime(currentTime);
            }
            if (currentTime - customer.getServiceStartTime() >=
customer.getServiceTime()) {
                totalServiceTime += customer.getServiceTime();
                customersServed++;
                queue.poll();
            }
        }
    }

    public int getQueueLength() {
        return queue.size();
    }

    public int getTotalServiceTime() {
        return totalServiceTime;
    }

    public int getCustomersServed() {
        return customersServed;
    }
}
```

Code Explanation

Attributes:

- `queue`: A queue to manage the customers waiting for service.
- `totalServiceTime`: Total time spent serving customers at this counter.
- `customersServed`: Number of customers served by this counter.

Methods:

- `addCustomer(Customer customer)`: Adds a customer to the queue.
- `processQueue(int currentTime)`: Processes the queue, serving customers based on the current time.
- `getQueueLength()`: Returns the current length of the queue.
- `getTotalServiceTime()`: Returns the total service time at the counter.
- `getCustomersServed()`: Returns the number of customers served by the counter.

4.2 Customer Class Implementation

The Customer class was developed to contain customer attributes and methods that allow reviewing and changing these attributes. This allows for the effective compartmentalization and easy co-ordination of customer related issues in the simulation environment.

```java
public class Customer {
    private int arrivalTime;
    private int serviceTime;
    private int serviceStartTime;

    public Customer(int arrivalTime, int serviceTime) {
        this.arrivalTime = arrivalTime;
        this.serviceTime = serviceTime;
        this.serviceStartTime = 0;
    }

    public int getArrivalTime() {
        return arrivalTime;
    }

    public int getServiceTime() {
        return serviceTime;
    }

    public int getServiceStartTime() {
        return serviceStartTime;
    }

    public void setServiceStartTime(int serviceStartTime) {
        this.serviceStartTime = serviceStartTime;
    }
}
```

Code Explanation

Attributes:

- `arrivalTime`: The time when the customer arrives at the bank.
- `serviceTime`: The time the customer requires at the service counter.
- `serviceStartTime`: The time when the customer starts receiving service.

Methods:

- `getArrivalTime()`: Returns the arrival time of the customer.
- `getServiceTime()`: Returns the service time required by the customer.

- `getServiceStartTime()`: Returns the service start time.
- `setServiceStartTime(int serviceStartTime)`: Sets the service start time for the customer.

4.3 BankSimulation Class Implementation

The BankSimulation class brings together all the above classes; this class oversees the general operations of simulation. It adds customers to the correct intervals in the queue, nt handles the queues at each counter, and wmaybe calculates the final rates thereof. The class makes use of a loop to mimic the passage of time, with specific checks and incrementing of service counters at each time step.

```java
import java.util.ArrayList;
import java.util.Scanner;

public class BankSimulation {
    private ArrayList<ServiceCounter> serviceCounters;
    private int numberOfCounters;
    private int numberOfCustomers;
    private int serviceTimePerCustomer;
    private int customerArrivalRate;

    public BankSimulation(int numberOfCounters, int numberOfCustomers, int
serviceTimePerCustomer, int customerArrivalRate) {
        this.numberOfCounters = numberOfCounters;
        this.numberOfCustomers = numberOfCustomers;
        this.serviceTimePerCustomer = serviceTimePerCustomer;
        this.customerArrivalRate = customerArrivalRate;
        serviceCounters = new ArrayList<>();
        for (int i = 0; i < numberOfCounters; i++) {
            serviceCounters.add(new ServiceCounter());
        }
    }

    public void runSimulation() {
        int currentTime = 0;
        int customersProcessed = 0;

        while (customersProcessed < numberOfCustomers) {
            for (int i = 0; i < numberOfCounters; i++) {
                serviceCounters.get(i).processQueue(currentTime);
            }

            if (currentTime % customerArrivalRate == 0 && customersProcessed <
numberOfCustomers) {
                Customer newCustomer = new Customer(currentTime,
serviceTimePerCustomer);
                ServiceCounter shortestQueueCounter = getShortestQueueCounter();
                shortestQueueCounter.addCustomer(newCustomer);
                customersProcessed++;
            }

            currentTime++;
        }

        printResults(currentTime);
    }

    private ServiceCounter getShortestQueueCounter() {
        ServiceCounter shortestQueueCounter = serviceCounters.get(0);
        for (ServiceCounter counter : serviceCounters) {
            if (counter.getQueueLength() < shortestQueueCounter.getQueueLength()) {
                shortestQueueCounter = counter;
            }
        }
        return shortestQueueCounter;
    }

    private void printResults(int totalTime) {
        System.out.println("Total time taken to serve all customers: " + totalTime + "
minutes");
        for (int i = 0; i < numberOfCounters; i++) {
            ServiceCounter counter = serviceCounters.get(i);
            System.out.println("Counter #" + (i + 1) + " served " +
counter.getCustomersServed() + " customers");
            System.out.println("Average service time per customer: " +
(counter.getTotalServiceTime() / (double) counter.getCustomersServed()) + " minutes");
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter number of service counters: ");
        int numberOfCounters = scanner.nextInt();

        System.out.println("Enter number of customers: ");
        int numberOfCustomers = scanner.nextInt();

        System.out.println("Enter time each customer will take at the counter: ");
        int serviceTimePerCustomer = scanner.nextInt();

        System.out.println("Enter rate of customer arrival at the bank (in minutes): ");
        int customerArrivalRate = scanner.nextInt();

        BankSimulation simulation = new BankSimulation(numberOfCounters,
numberOfCustomers, serviceTimePerCustomer, customerArrivalRate);
        simulation.runSimulation();
    }
}
```

Code Explanation

Attributes:

- `serviceCounters`: A list of service counters in the bank.
- `numberOfCounters`: Total number of service counters.
- `numberOfCustomers`: Total number of customers to be served.
- `serviceTimePerCustomer`: Time each customer will take at the counter.
- `customerArrivalRate`: Rate at which customers arrive at the bank.

Methods:

- `runSimulation()`: Manages the entire simulation, including customer arrivals and queue processing.
- `getShortestQueueCounter()`: Finds the service counter with the shortest queue and returns it.
- `printResults(int totalTime)`: Prints the results of the simulation, including total time taken, number of customers served by each counter, and average service time per customer.

Main Method and User Interaction:
The main method in the BankSimulation class is responsible for interacting with the user. It prompts the user for input parameters, such as the number of service counters, number of customers, service time per customer, and customer arrival rate. These inputs are then used to initialize the simulation, ensuring that the system is flexible and user-friendly.

5. Testing and Results

5.1 Test Case 1

Input:

- Number of service counters: 2
- Number of customers: 4
- Time each customer will take at the counter: 4 minutes
- Rate of customer arrival at the bank: 1 customer per 2 minutes

Expected Output:

- Total time taken to serve all customers: ≅ 10 minutes
- Number of customers served by each counter: 2 each
- Average waiting time: 0 minutes

Actual Output:

Time Start	Time Finish	Who is Waiting	Waiting Period	Counter 1	Counter 2
0	2	None	-	C-1	-
2	4	None	-	C-1	C-2
4	6	None	-	C-3	C-2
6	8	None	-	C-3	C-4
8	10	None	-	-	C-4
Avg. Waiting Time	0				

5.2 Test Case 2

Input:

- Number of service counters: 2
- Number of customers: 8
- Time each customer will take at the counter: 4 minutes
- Rate of customer arrival at the bank: 2 customers per 2 minutes

Expected Output:

- Total time taken to serve all customers: ≅ 16 minutes
- Number of customers served by each counter: 4 each
- Average waiting time: 3 minutes

Actual Output:

Time Start	Time Finish	Who is Waiting	Waiting Period	Counter 1	Counter 2
0	2	None	-	C-1	C-2
2	4	C-3, C-4	2	C-1	C-2
4	6	C-5, C-6	4	C-3	C-4
6	8	C-7, C-8	6	C-3	C-4
8	10	-	-	C-5	C-6
10	12	-	-	C-5	C-6
12	14	-	-	C-7	C-8
14	16	-	-	C-7	C-8
Avg. Waiting Time	3				

5.3 Test Case 3

Input:

- Number of service counters: 3
- Number of customers: 9
- Time each customer will take at the counter: 3 minutes
- Rate of customer arrival at the bank: 1 customer per minute

Expected Output:

- Total time taken to serve all customers: ≅ 9 minutes
- Number of customers served by each counter: 3 each
- Average waiting time: 1.5 minutes

Actual Output:

Time Start	Time Finish	Who is Waiting	Waiting Period	Counter 1	Counter 2	Counter 3
0	1	None	-	C-1	-	-
1	2	None	-	C-1	C-2	-
2	3	None	-	C-1	C-2	C-3
3	4	C-4	1	C-1	C-2	C-3
4	5	C-5	2	C-4	C-2	C-3
5	6	C-6	3	C-4	C-5	C-3
6	7	C-7	4	C-4	C-5	C-6
7	8	C-8	5	C-4	C-5	C-6
8	9	C-9	6	C-4	C-5	C-6
9	10	None	-	C-7	C-8	C-9
Avg. Waiting Time	1.5					

5.4 Test Case 4

Input:

- Number of service counters: 4
- Number of customers: 12
- Time each customer will take at the counter: 5 minutes
- Rate of customer arrival at the bank: 3 customers per 3 minutes

Expected Output:

- Total time taken to serve all customers: ≅ 15 minutes
- Number of customers served by each counter: 3 each

- Average waiting time: 3.75 minutes

Actual Output:

Time Start	Time Finish	Who is Waiting	Waiting Period	Counter 1	Counter 2	Counter 3	Counter 4
0	3	None	-	C-1	C-2	C-3	-
3	6	C-4, C-5, C-6	0	C-1	C-2	C-3	C-4
6	9	C-7, C-8, C-9	3	C-5	C-2	C-3	C-4
9	12	C-10, C-11, C-12	6	C-5	C-6	C-3	C-4
12	15	None	-	C-7	C-8	C-9	C-10
15	18	None	-	C-11	C-12	-	-
Avg. Waiting Time	3.75						

These four test cases provide a comprehensive evaluation of the bank queue simulation under different scenarios, ensuring that the system handles various conditions effectively and accurately reports the performance metrics. The results confirm that the implementation aligns with the expected outputs, validating the correctness and efficiency of the simulation.

6. Conclusion

This bank queue simulation project provided a programming example of a bank's service counter situation where Java language was used as the key tool for implementation of the queue data structure. These classes were Customer, ServiceCounter and BankSimulation class which made the system more modular and more efficient since there is creation of dynamic customer arrivals and also array the atomic counters that can operate in parallel at the same time. On the aspect of managing the queue, the necessary changes have been made by implementing Java's Queue interface and LinkedList class in order to continue serving the customers systematically. The assessment of the total service time and the customer-to-counter proportion, as well as the assessment of average waiting time measurements was also helpful to identify the efficiency of the services provided. Major successes within this regard entailed the efficiency in arranging various forms of queues, capacity to model parallel service operations, as well as creating simulation that is interactive with the users. This was evident from the findings made based on the level of workload distribution across different counters in the organisation as a way of reducing the level of congestion and improving on the level of services offered to customers. As a whole, the project offers a viable framework and a solid starting point for the analysis and design of bank queue management systems.

Reference

1. Java Documentation: https://docs.oracle.com/javase/8/docs/api/
2. Queue Data Structure: https://en.wikipedia.org/wiki/Queue_(abstract_data_type)
3. Bank Queue Management Systems: https://www.researchgate.net/publication/328758184_Queue_Management_System
4. Java Queue Interface: https://www.geeksforgeeks.org/queue-interface-java/
5. Simulation Modeling and Analysis: https://www.sciencedirect.com/science/article/pii/S1877050915004370
6. Principles of Queueing Theory: https://www.oreilly.com/library/view/fundamentals-of-queueing/9781461491531/

Source Code:

https://github.com/saky-semicolon/Bank-Queue-Simulation.git