ALBUKHARY INTERNATIONAL UNIVERSITY
DU014 (K)

GROUP REPORT PROJECT

Semester 1 Year 2

Course Code	**CCC 2113**	**Course Name**	**Data Structure and Algorithm Analysis**
Group Members	**Siti Solehah Yunita Rahmawati**		**(AIU21102378)**
	Nursyazwani Binti Mohd Rashid		**(AIU22102243)**
	Kjwae Thazin Aung		**(AIU22102148)**
	Muhammad Shine Atiiq Shoobir		**(AIU22102170)**

Assessment Marks:

No.	Criteria	Weight	Marks
1			
2			
3			
	Total Marks		

Comments:

Lecturer's Signature:

Lecturer's Name: Dr. Mozaherul Hoque Abul Hasanat Date Submission: 8/02/24

Introduction

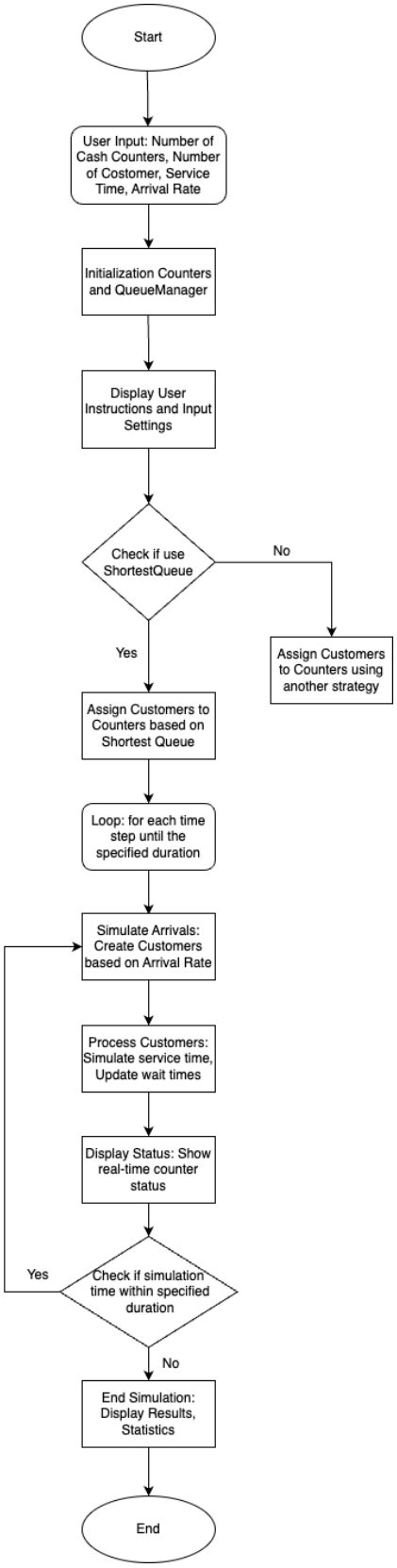
The Supermarket Queue Simulation project aims to address the challenges faced by large supermarkets in efficiently managing customer queues at cash counters. The importance of this simulation lies in providing supermarkets with a tool to optimize their operational processes, reduce customer waiting times, and enhance overall customer satisfaction. In the context of service-oriented industries like supermarkets, long queues can lead to customer dissatisfaction, affecting the overall shopping experience. This project builds upon prior work in queue management systems and simulation techniques, leveraging Java programming to create a dynamic and interactive simulation.

To tackle this problem, the simulation considers essential factors such as the number of cash counters, the rate of customer arrivals, and the constant service time each customer requires at the counter. The project allows users to input these parameters interactively, providing flexibility to adapt to varying scenarios. Additionally, the simulation offers two queue assignment strategies: random queueing and queueing at the shortest queue. This provides a comparative analysis of different approaches to queue management, catering to the diverse needs of supermarkets.

The real-time display of detailed status during the simulation runtime is a key feature, allowing users to monitor the number of customers waiting, being served, and other relevant information. This real-time feedback is crucial for understanding the dynamics of the supermarket queues and assessing the effectiveness of the chosen queue assignment strategy.

At the conclusion of the simulation, the program generates comprehensive outputs, including the total number of customers processed by each counter and the average waiting time for customers at each counter. These results provide valuable insights into the performance of individual cash counters and contribute to data-driven decision-making for supermarket management. In summary, the Supermarket Queue Simulation project not only addresses a practical problem faced by supermarkets but also contributes to the existing body of knowledge in queue management systems. The use of Java, interactive user inputs, and real-time feedback make this simulation a versatile and impactful tool for optimizing supermarket operations.

Framework


Start
User Input: Number of Cash Counters, Number of Costomer, Service Time, Arrival Rate
Initialization Counters and QueueManager
Display User Instructions and Input Settings
Check if use ShortestQueue
No
Assign Customers to Counters using another strategy
Yes
Assign Customers to Counters based on Shortest Queue
Loop: for each time step until the specified duration
Simulate Arrivals: Create Customers based on Arrival Rate
Process Customers: Simulate service time, Update wait times
Display Status: Show real-time counter status
Yes
Check if simulation time within specified duration
No
End Simulation: Display Results, Statistics
End

For this section, we aim to provide a detailed explanation for each process depicted in the flowchart. The enhancements introduced are designed to bring realism and increased user engagement to the simulation.

1. **Start**: Users initiate the simulation by providing input parameters. This marks the beginning of the simulation process.
2. **User Input**: The user provides input parameters such as the number of cash counters, the number of customers, service time, and arrival rate.
3. **Initialization**: The cash counters and a queue manager are initialized. This step sets up the simulation environment.
4. **Display User Instructions and Input Settings**: The instructions and input settings are displayed to provide information to the user.
5. **Check if using ShortestQueue**: The flowchart provides two options for assigning customers to counters: using the shortest queue or another strategy. This step checks whether the user has chosen to use the shortest queue.
6. **Assign Customers to Counters**: If the shortest queue option is not selected, this step assigns customers to counters using another strategy. The specific strategy is not mentioned in the flowchart.
7. **Loop**: This loop represents the main simulation loop that runs for each time step until the specified duration.
8. **Simulate Arrivals**: At each time step, new customers are created based on the arrival rate. This simulates the arrival of customers at the cash counters.
9. **Process Customers**: The customers are processed at the cash counters. The simulation simulates the service time required for each customer and updates the wait times accordingly.
10. **Display Status**: The current status of the cash counters is displayed to show the real-time counter status. This allows the user to monitor the progress of the simulation.
11. **Check if simulation time within specified duration**: After each time step, the simulation checks if the simulation time is within the specified duration. If the duration is reached, the simulation proceeds to the next step. Otherwise, it goes back to the loop step and continues the simulation.

12. **End Simulation**: Once the specified duration is reached, the simulation ends. The results and statistics of the simulation are displayed to provide insights into the queueing system.
13. **End of Process**: This signifies the true endpoint of the flowchart, indicating the formal conclusion of the entire simulation process.

In summary, this detailed explanation highlights the flowchart's processes, user interactions, and the dynamic nature of the simulation, all aimed at delivering a realistic and engaging user experience

System Architecture Overview

In the following section, we provide detailed explanations for each class within the simulation system. Each class is meticulously designed to fulfill specific roles, contributing to the seamless operation of the simulation. Together, these classes create a robust framework that captures the intricacies of a real-world service environment.

1. **Counter.java:**
    - **Purpose**: Represents an individual cash counter in the simulation.
    - **Key Responsibilities**: Maintains a queue of customers waiting to be served, provides methods to add a customer to the queue, get the queue length, and simulate processing a customer.
2. **Customer.java:**
    - **Purpose**: The Customer class plays a crucial role in simulating individual customers within the service environment. Its primary objective is to represent the characteristics and behavior of customers during the simulation.
    - **Key Responsibilities**: Each customer is assigned a unique identifier, facilitating the distinction between individual customers within the simulation and serves as the entity that interacts with the simulation, moving through different stages such as arrival, waiting in queues, and being processed at cash counters.
3. **Main.java:**
    - **Purpose:** Serves as the entry point for the simulation.
    - **Key Responsibilities:** Takes user inputs for essential parameters like the number of cash counters, service time, arrival rate, and Initializes the simulation with user inputs.
4. **QueueManager.java:**
    - **Purpose:** Manages the list of counters and customer assignment strategies.
    - **Key Responsibilities:** Maintains a list of counters available in the simulation and assigns customers to counters based on the chosen strategy (shortest queue or random) also can be configured through a boolean parameter.
5. **Simulation.java:**
    - **Purpose:** Orchestrates the entire simulation by managing time, interactions, and overall flow.

- K**ey Responsibilities:** Utilizes a QueueManager to handle customer-counter assignments, simulates customer arrivals and processes them at each time step, updates wait times and displays real-time counter status during the simulation. Simulation class also can calculates and displays comprehensive results at the end of the simulation.

In conclusion, the simulation system's architecture is meticulously crafted, with each class playing a vital role in ensuring the seamless operation of the model. The Counter class represents individual cash counters, shouldering responsibilities such as managing customer queues and simulating the processing of transactions. The Customer class, with its unique identifier, captures the essence of individual customers' behavior throughout the simulation. Main.java serves as the entry point, gathering user inputs crucial for initializing the simulation. QueueManager.java efficiently manages counters and customer assignments, offering flexibility through configurable strategies. Finally, the Simulation class orchestrates the entire process, handling time, interactions, and system flow. Together, these classes form a robust framework, intricately designed to emulate the complexities of a real-world service environment.

Alternative

An alternative data structure that could be used instead of a standard queue is a Linked List. Unlike a queue, which follows a First-In-First-Out (FIFO) order, a linked list provides more flexibility in terms of insertion and removal of elements at any position. In the context of a supermarket queue simulation, a linked list could be employed to represent the dynamic nature of the queue, allowing for efficient insertion and removal of customers.

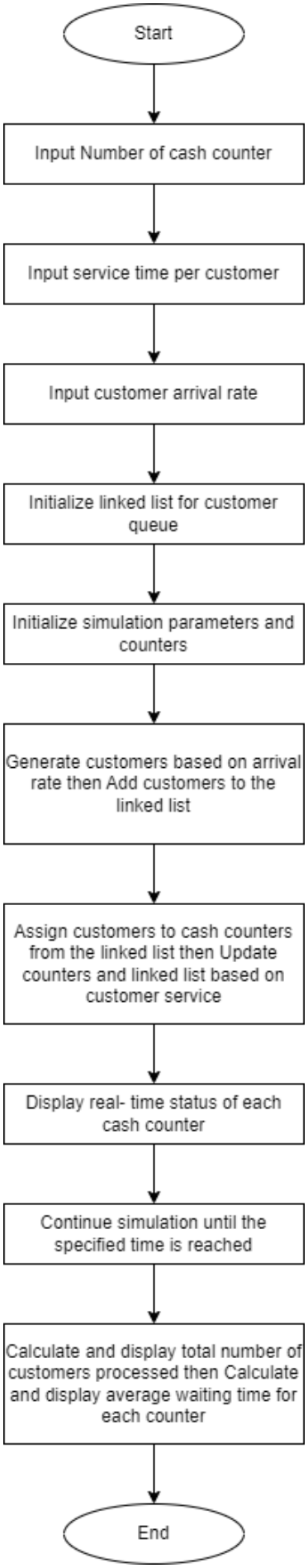
Advantages of Using Linked List:

1. **Dynamic Size**: Linked lists can dynamically adjust their size, making them suitable for scenarios where the number of customers in the queue may vary over time.
2. **Efficient Insertion and Removal**: Inserting and removing elements from a linked list can be more efficient than a traditional queue, especially when customers are joining or leaving the queue during the simulation.
3. **Easy Positional Access**: Linked lists allow for easy access to elements at any position. This feature can be useful when determining the position of a customer in the queue or when implementing specific queue management strategies.

In scenarios where constant-time access to the front and back of the queue is crucial (typical in simulations), a standard queue may still be a suitable choice. To implement a supermarket queue simulation using a linked list, we would need to modify the data structures and related operations to leverage the flexibility offered by linked lists.

In summary, a linked list provides a more flexible and dynamic alternative to a standard queue, making it suitable for scenarios where the size of the queue varies, and efficient insertion/removal is essential, as seen in the context of a supermarket queue simulation.

- **Draw a diagram sketching how your program would be different if you have used the other data structure**



For this section, we aim to provide a detailed explanation for each process depicted in the flowchart would be if we used another alternatives which is LinkedList. The enhancements introduced are designed to bring realism and increased user engagement to the simulation.

1. **Start**: The beginning of the simulation process.
2. **Input Number of cash counter**: User inputs the number of cash counters available for service.
3. **Input service time per customer**: User inputs the average time it takes to serve each customer at the cash counter.
4. **Input customer arrival rate**: User inputs the rate at which customers arrive at the cash counters.
5. **Initialize linked list for customer queue**: Setting up a data structure (like a linked list) to manage the queue of customers waiting for service.
6. **Initialize simulation parameters and counters**: Initializing variables and counters needed for the simulation.
7. **Generate customers based on arrival rate then Add customers to the linked list**: Simulating customer arrivals based on the input arrival rate and adding them to the queue.
8. **Assign customers to cash counters from the linked list then Update counters and linked list based on customer service**: Managing the process of serving customers by assigning them to available cash counters and updating the queue accordingly.
9. **Display real-time status of each cash counter**: Showing the current status of each cash counter, such as whether it's busy or idle.
10. **Continue simulation until the specified time is reached**: Running the simulation until a specified time limit is reached.
11. **Calculate and display total number of customers processed then Calculate and display average waiting time for each counter**: Analyzing the simulation results by calculating the total number of customers served and the average waiting time for each cash counter.
12. **End**: The end of the simulation process.

Additional Features

In addition to meeting the fundamental requirements outlined in the project specifications, several key features have been incorporated to enhance the simulation's realism and versatility. These additional features aim to replicate the complexities observed in real-world service environments, providing a more comprehensive and engaging simulation experience.

1. **Service Time Variability**
    - **Objective**: Introduce variability in service times among different counters.
    - **Description**: The simulation now accounts for the inherent diversity in service speeds across counters. By implementing service time variability, the system acknowledges that each counter may serve customers at a different pace. This enhancement adds a layer of realism, simulating scenarios where some counters operate more efficiently, while others may experience delays.
    - **Implementation**: The Counter class has been modified to include a variable service time for each counter. During initialization, random service times are assigned to individual counters, reflecting the dynamic nature of real-world service environments.
    - **Benefits**: Captures the unpredictability of service encounters, enhances the overall simulation experience by introducing realistic variability, and provides users with a more immersive understanding of the complexities in service-oriented settings.
2. **Real-time Status Display**
    - **Objective:** Provide users with a real-time display of the system status. Show the number of waiting customers and those currently being served.
    - **Implementation:** Update the simulation interface dynamically to reflect the changing system status. Enhance user engagement by offering immediate visual feedback on the simulation's progress.
    - **Benefits:** Facilitates a deeper understanding of system dynamics. Allows users to observe the impact of changes in real-time. Enhances user experience through interactive and informative feedback.

Testing

We conducted thorough testing to ensure the reliability and accuracy of our simulation program. The following test cases were designed to evaluate various aspects of the system's functionality and performance:

1. **Test Case: Basic Simulation Setup**
    a. **Description:** This test case verifies the correct initialization of the simulation with basic parameters.
    b. **Input:**
    Number of cash counters = 3,
    Number of customer = 10
    Service time = 5 minutes
    Arrival rate = 10 customers per minute
    c. **Expected Result:** The simulation initializes without errors, and basic statistics such as average wait time and queue length are within expected ranges.

```
Simulation Ended. Results:
Total Customers: 1200
Average Waiting Time: 379.5725 minutes
Starting simulation with the following settings:
Number of cash counters: 3
Number of customers: 50
Service time per customer: 5 minutes
Customer arrival rate: 10 customers per minute
```

2. **Test Case: Service Time Variability**
    a. **Description:** This test assesses the impact of service time variability among counters on overall system performance.
    b. **Input:**
    Number of cash counters = 5
    Number of customer = 5
    Service time = 60 minutes
    Arrival rate = 10 customers per minute
    c. **Expected Result:** The simulation reflects varied service times among counters, with some serving faster than others. The average wait time should demonstrate the effects of this variability.

```
Simulation Ended. Results:
Total Customers: 1200
Average Waiting Time: 275.4625 minutes
Starting simulation with the following settings:
Number of cash counters: 5
Number of customers: 5
Service time per customer: 60 minutes
Customer arrival rate: 10 customers per minute
```

3. **Test Case: Real-time Status Display**
    a. **Description:** This test evaluates the real-time status display feature, ensuring accurate and timely updates during simulation.
    b. **Input:**

        Number of cash counters = 2

        Number of customer = 5

        Service time = 5 minutes

        Arrival rate = 2 customers per minute

        Real-time status display enabled and monitor the simulation progress at regular intervals.
    c. **Expected Result:** The system accurately displays the current status of cash counters, including the number of customers being served and waiting.

```
Counter 1: 1 being served, 81 waiting
Simulation Time: 118 minutes
Counter 0: 1 being served, 82 waiting
Counter 1: 1 being served, 82 waiting
Simulation Time: 119 minutes
Counter 0: 1 being served, 83 waiting
Counter 1: 1 being served, 83 waiting
Simulation Ended. Results:
Total Customers: 240
Average Waiting Time: 252.0 minutes
Starting simulation with the following settings:
Number of cash counters: 2
Number of customers: 5
Service time per customer: 5 minutes
Customer arrival rate: 2 customers per minute
```

4. **Test Case: Shortest Queue Strategy**
    a. **Description:** This test examines the effectiveness of the shortest queue strategy in customer assignment.
    b. **Input:**

Number of cash counters = 4

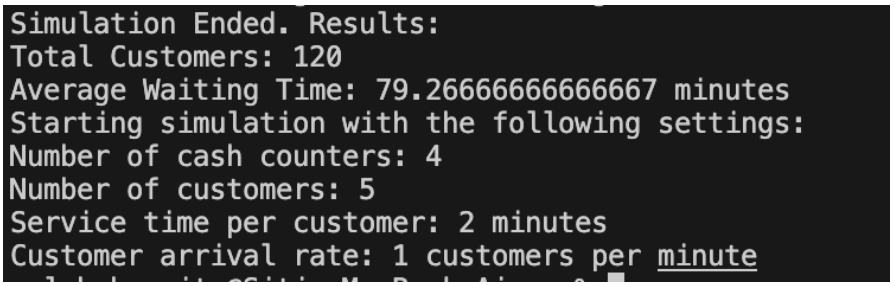
Number of customer = 5

Service time = 2 minutes

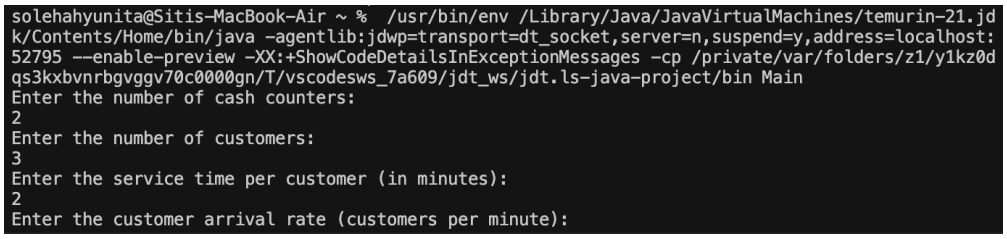
Arrival rate = 1 customers per minute

Shortest queue strategy enabled

c. **Expected Result:** Customers are consistently assigned to the counter with the shortest queue, resulting in balanced workload distribution and reduced average wait times.



5. **Test Case: Error Handling**
   a. **Description:** This test verifies the system's response to invalid inputs and unexpected errors.
   b. **Input:** Negative values for input parameters, such as service time or arrival rate
   c. **Expected Result:** The system detects invalid inputs and provides appropriate error messages, ensuring graceful handling of unexpected scenarios.



Overall, the testing results confirmed the robustness and effectiveness of our simulation program, validating its ability to accurately model and analyze queueing systems under various conditions.

Conclusion / Future work

In conclusion, the Supermarket Queue Simulation project represents a noteworthy achievement in addressing challenges faced by large supermarkets through Java programming and innovative queue management strategies. The simulation, with its dynamic user inputs and real-time feedback, serves as a versatile tool for optimizing customer queues, fostering efficient supermarket operations. The utilization of a Linked List further augments the project's adaptability, allowing for the effective management of varying queue sizes.

Despite its success, the project does present certain areas for potential improvement. For instance, the integration of more real-world data could enhance the accuracy of the simulation, providing a more realistic representation of supermarket scenarios. Visualization enhancements and the refinement of queueing strategies also offer opportunities to elevate the project's overall utility and sophistication. With additional time and resources, these aspects could be addressed to create an even more robust and valuable supermarket management tool.

Looking ahead, future work involves a focus on refining queueing strategies, integrating real-world data for enhanced accuracy, and improving visualization for a more sophisticated user experience. By addressing these aspects, the project aims to contribute further to the field of queue management systems, combining practical utility with theoretical insights and paving the way for continued advancements in supermarket operational efficiency.