

DOCUMENTATION

ASSIGNMENT 2

- QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS-

STUDENT NAME: Pitaru Alexandra-Elena
GROUP: 30423

CONTENTS

| | |
|--|----|
| 1. Assignment Objective | 3 |
| 2. Problem Analysis, Modeling, Scenarios, Use Cases..... | 4 |
| 3. Design | 4 |
| 4. Implementation | 6 |
| 5. Results..... | 11 |
| 6. Conclusions..... | 11 |
| 7. Bibliography | 12 |

1. Assignment Objective

2. Main objective of the assignment:

- Design and implement an application aiming to analyze queuing-based systems by (1) simulating a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues, and (2) computing the average waiting time, average service time and peak hour .

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e., more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier.

The queues management application should simulate (by defining a simulation time *tsimulation*) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), *tarrival* (simulation time when they are ready to enter the queue) and *tservice* (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue). The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with the minimum waiting time when its *tarrival* time is greater than or equal to the simulation time ($tarrival \geq tsimulation$).

3. The sub-objectives:

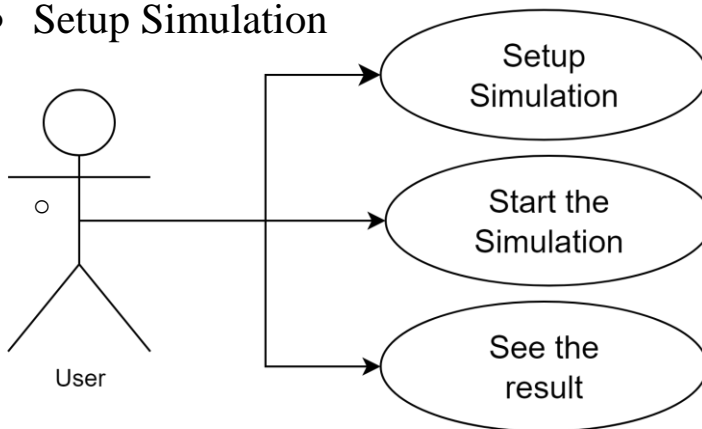
- a. Analyze the problem and identify requirements (Section 2)
 - i. Gather user requirements
 - ii. Generate random clients
 - iii. Identify edge cases
- b. Design the simulation application (Section 3)
 - i. User interface design
 - ii. Algorithm design
 - iii. Data model design
 - iv. Error handling design
- c. Implement the simulation application (Section 4)
 - i. Code for user interface components
 - ii. Implement back-end logic
 - iii. Implement startegies
 - iv. Implement error handling
- d. Test the polynomial calculator (Section 5)

4. Problem Analysis, Modeling, Scenarios, Use Cases

- Functional requirements:

- The simulation application should allow users to setup the simulation
- The simulation application should allow users to start the simulation
- The simulation application should display the real-time queues evolution
- The simulation application should display results(average waiting time, average service time, peak hour).

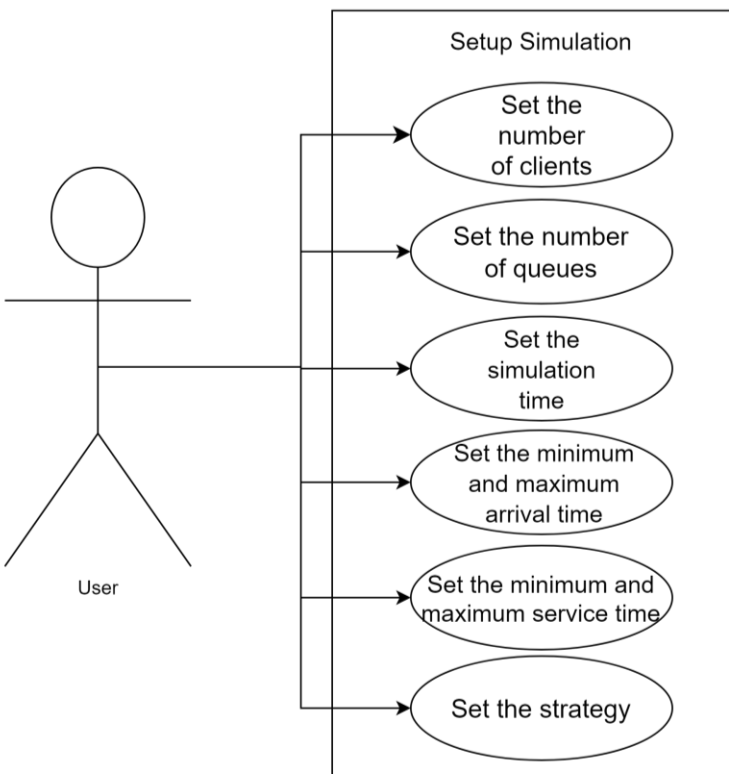
- Setup Simulation



Actor: User

Steps:

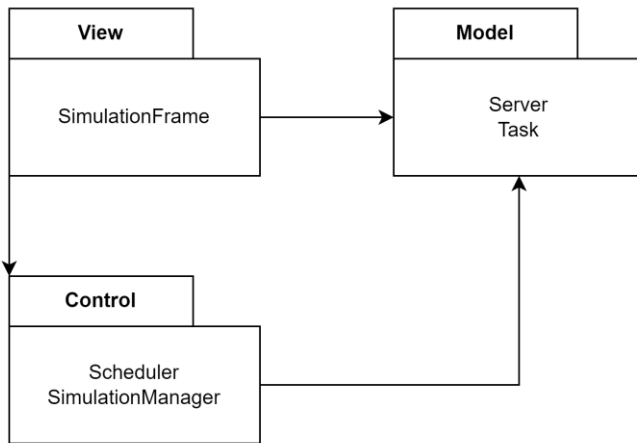
- User opens the simulation app
- User selects the text fields to input data



- Non-Functional Requirements:

- Usability: The simulation application should be intuitive and easy to use by the user
- User Interface Design: The simulation application's user interface should be visually appealing, with clear and explicit labeling of buttons and controls.
- Performance: The application should be responsive and performant, even when handling large datasets or running complex simulations.

5. Design



1. The Control-Model-View (CMV)

architectural pattern is a design paradigm that separates the concerns of an application into three distinct components: Control, Model, and View.

- **Model (M):**

In the queue management simulation application, the Model represents the core data structures and algorithms used to simulate the behavior of servers, tasks, and

queues. It encapsulates the logic for task generation, server scheduling, and queue management. Classes such as Server, Task, and Scheduler in the model package fulfill this role. They define the underlying structure and behavior associated with simulating queue management scenarios.

- **View (V):**

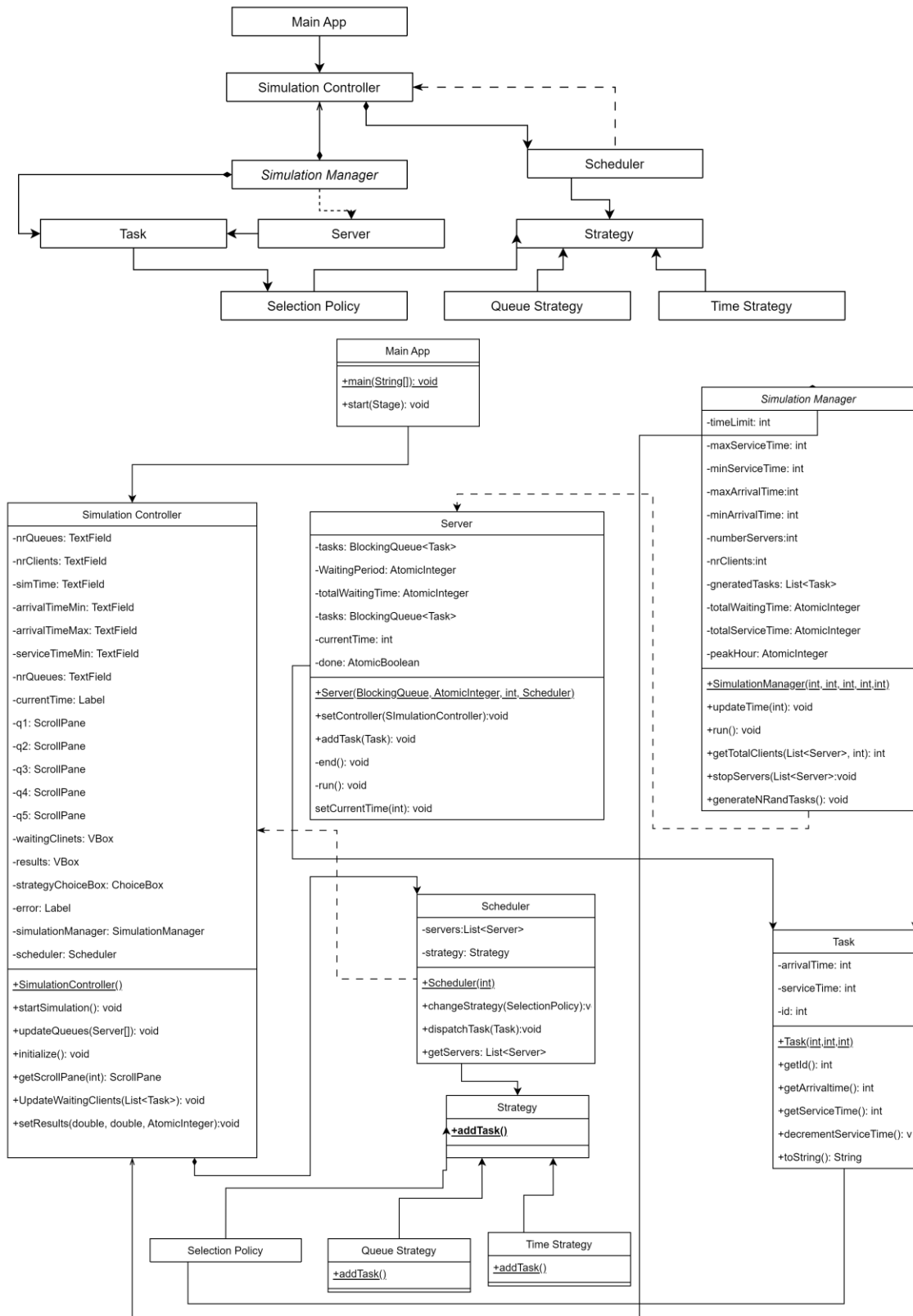
The View component is responsible for presenting the simulation's status and visualizing the queue evolution to the user. It translates the simulation data provided by the Model into a user-friendly format and displays it in a graphical user interface (GUI). In the queue management simulation application, the View class in the view package represents the GUI where users interact with the simulation. It displays real-time information about servers, tasks, and queues and provides controls for starting, pausing, and stopping the simulation.

- **Control (C):**

The Control acts as an intermediary between the Model and View components, coordinating user interactions, retrieving data from the Model, and updating the View accordingly. It contains the application's logic for initializing the simulation, processing user inputs, and controlling the simulation flow. In the queue management simulation application, the SimulationManager class in the control package serves as the Control component. It orchestrates the simulation process, manages task dispatching, and updates the View with simulation results and real-time queue evolution.

By adhering to the MVC pattern in the queue management simulation application, I have achieved a clear separation of concerns, enhancing the code's readability, maintainability, and extensibility. Each component focuses on its specific responsibilities, promoting code reusability and facilitating future enhancements or modifications to the application. By adopting the CMV pattern in my project, I have achieved a clear separation of concerns, making my code easier to understand, maintain, and extend. Each component focuses on a specific aspect of the application, promoting code reusability and modularity.

2. Class Diagram



3. Used data structures

In the queue management system project, a variety of data structures are employed to represent and manage the simulation's elements efficiently. These include primitive data types such as integers and doubles, as well as more complex data structures like HashMap, ArrayList, and arrays.

- **HashMap:** Used to store the tasks' arrival times and service times, facilitating efficient retrieval and management of task data.
- **ArrayList:** Utilized to store generated tasks, allowing for dynamic resizing and easy manipulation of task lists during the simulation.
- **Arrays:** Employed in various contexts, such as representing server queues and managing server instances.

Additionally, custom objects such as Server, Task, and Scheduler are created and utilized to encapsulate related data and behaviors, enhancing modularity and maintainability.

4. Used Algorithms

a. Task Dispatching:

- The algorithm for task dispatching involves assigning arriving tasks to servers based on certain criteria (e.g., shortest queue length or round-robin scheduling).

b. Queue Management:

- Algorithms are used to manage server queues, including enqueueing and dequeueing tasks, as well as determining queue lengths and statuses.

c. Simulation Control:

- Algorithms control the simulation flow, including starting, pausing, and stopping the simulation, as well as updating the simulation state in real-time.

d. Performance Metrics Calculation:

- Algorithms calculate performance metrics such as average waiting time, peak hour, and server utilization based on collected simulation data.

f. Time Management:

- Algorithms manage simulation time, including advancing the simulation clock, handling task arrival times, and tracking task service times.

These algorithms collectively enable the simulation system to accurately model and analyze queue management scenarios, providing valuable insights into system performance and efficiency.

6. Implementation

I. Class Description

1) The Model:

a. Task Class:

The Task class represents a task in the queue management system and contains attributes such as arrival time, service time, and unique identifier.

Constructor:

public Task(int arrivalTime, int serviceTime, int id): Initializes a task with the specified arrival time, service time, and unique identifier.

Methods:

- public int getArrivalTime(): Returns the arrival time of the task.
- public int getServiceTime(): Returns the service time required for the task.
- public int getId(): Returns the unique identifier of the task.

b. Server Class:

The Server class represents a server in the queue management system and handles task processing.

Constructor:

public Server(int id): Initializes a server with the specified unique identifier.

Methods:

- public void setCurrentTime(int currentTime): Sets the current simulation time for the server.
- public void end(): Signals the server to stop processing tasks.
- Other methods related to task processing and queue management.

2) The Control:

a) Scheduler Class:

The Scheduler class manages the assignment of tasks to servers based on specified scheduling policies.

Constructor:

public Scheduler(int numberOfServers): Initializes the scheduler with the specified number of servers.

Methods:

- public void dispatchTask(Task task): Assigns a task to an available server based on the selected scheduling policy.
- Other methods related to server management and scheduling policies

b) SimulationManager Class:

The SimulationManager class controls the simulation process, including task generation, server management, and performance metric calculation.

Constructor:

public SimulationManager(int timeLimit, int maxServiceTime, int minServiceTime, int maxArrivalTime, int minArrivalTime, int numberOfServers, int numberOfClients, SimulationController controller):
Initializes the simulation manager with simulation parameters and a reference to the controller.

Methods:

- public void run(): Starts the simulation process.
- Other methods for updating simulation time, stopping servers, and calculating performance metrics.

3) The View:

a) SimulationController Class:

The SimulationController class handles user interactions and updates the graphical user interface (GUI) with real-time simulation data.

Constructor:

public SimulationController(): Initializes the controller and sets up the GUI components.

Methods:

- public void setCurrentTimeLabel(int currentTime): Updates the current time label in the GUI.
- Other methods for updating queue visuals, displaying simulation results, and handling user inputs.

4) The Application:

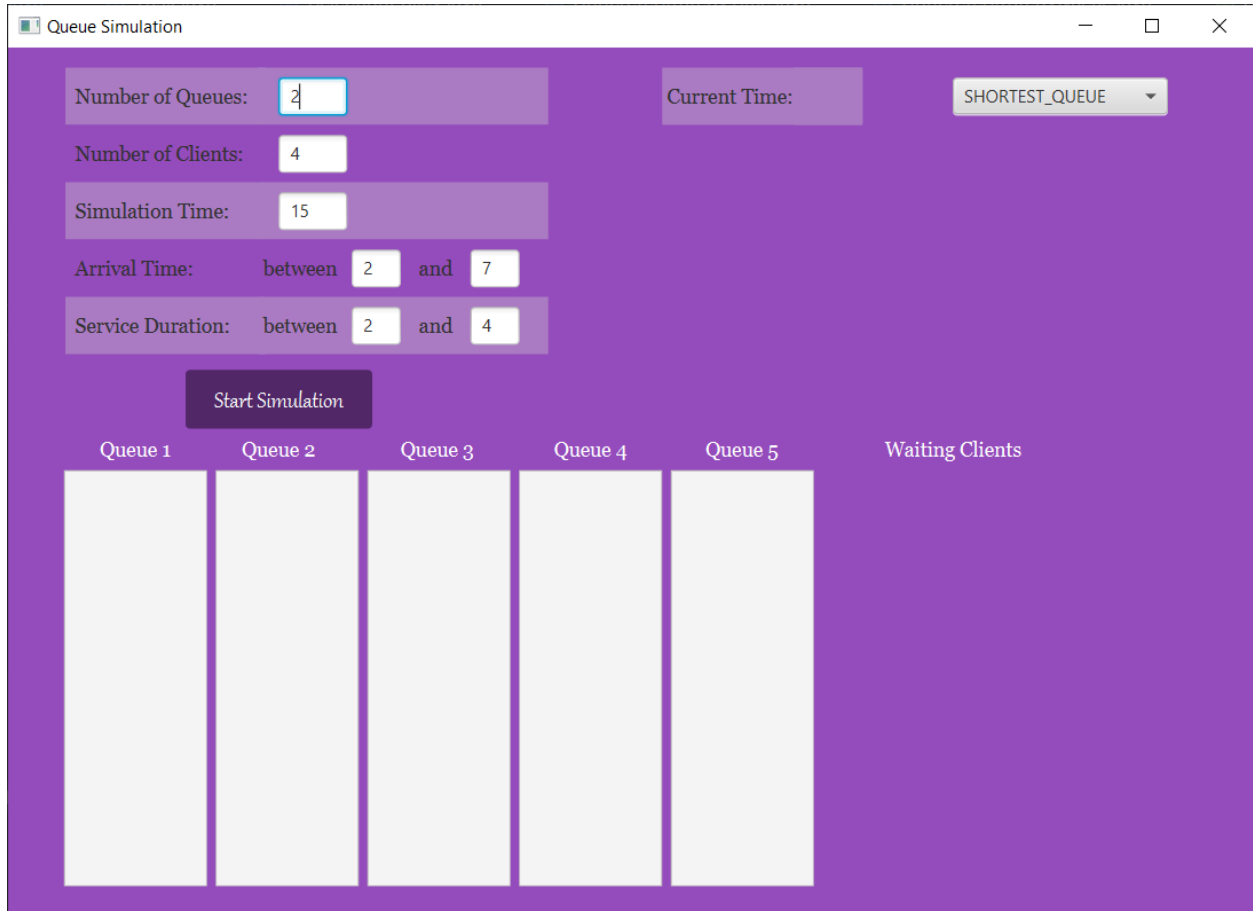
a) Main Class:

The Main class serves as the entry point of the queue management system application..

Main Method:

- public static void main(String[] args): Instantiates the SimulationView and SimulationController classes to start the application and GUI.

II. Graphical User Interface description



The screenshot shows a web-based GUI for a queue simulation. The interface has a purple background. At the top, there's a title bar 'Queue Simulation' with standard window controls. Below this, there are several input fields and a dropdown menu. The 'Number of Queues' field is set to 2. The 'Number of Clients' field is set to 4. The 'Simulation Time' field is set to 15. The 'Arrival Time' field is set to 'between 2 and 7'. The 'Service Duration' field is set to 'between 2 and 4'. The 'Current Time' field is empty. The 'Queue Selection' dropdown menu is set to 'SHORTEST_QUEUE'. Below these fields is a 'Start Simulation' button. At the bottom, there are five vertical bars representing 'Queue 1', 'Queue 2', 'Queue 3', 'Queue 4', and 'Queue 5'. To the right of these bars is a section labeled 'Waiting Clients'.

The Graphical User Interface (GUI) serves as the bridge between the user and the queue management system application, facilitating interaction and providing feedback.

The interface is designed to be intuitive and user-friendly, ensuring ease of use for individuals of varying technical backgrounds. Here's how the interface functions:

Input Fields: Users can input the parameters required for the simulation, such as time limits, service times, arrival times, and the number of servers and clients, into designated text fields.

Simulation Controls: Buttons allow users to start the simulation as needed. These controls provide users with the ability to manage the simulation process effectively.

Real-time Visualization: The interface dynamically displays the evolution of queues and server states in real-time. This visualization enables users to monitor the simulation's progress and make informed decisions.

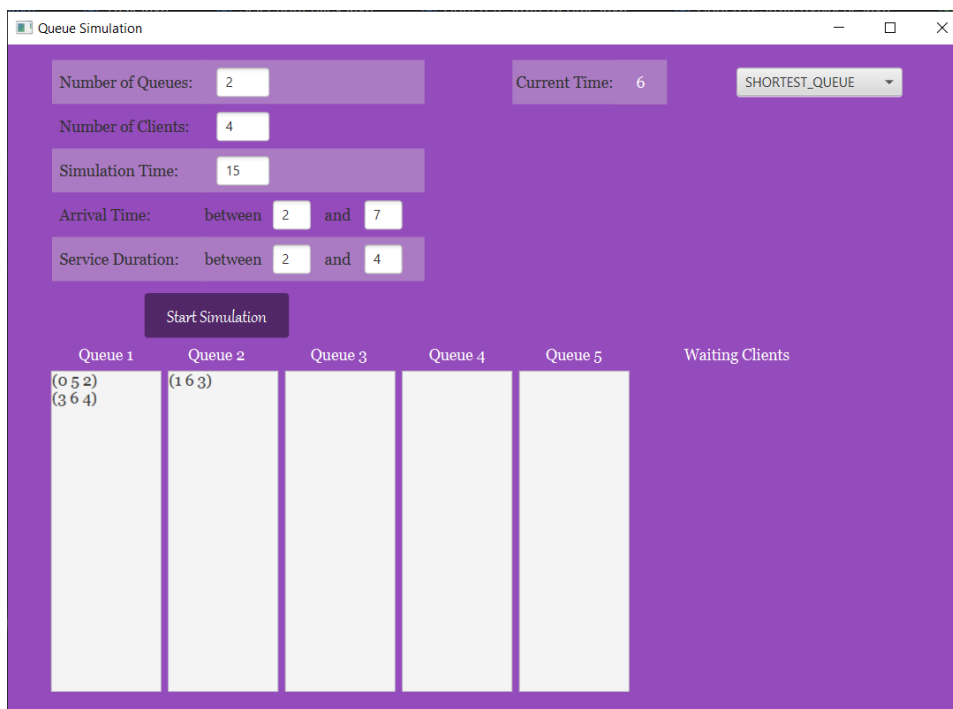
Result Display: Performance metrics, including average waiting time and peak hour, are prominently displayed on the interface. This feedback mechanism keeps users informed about the simulation's outcomes and helps them assess its effectiveness.

Error Handling: The interface includes error handling mechanisms to ensure that user inputs are valid and adhere to specified formats. Clear error messages guide users in correcting any input mistakes and prevent simulation errors.

In summary, the GUI of the queue management system application is designed to streamline user interaction, provide real-time feedback, and ensure a smooth simulation experience.

7. Results

I sent the logs of three test simulation into the text files found in the project resources.



The screenshot shows a Java Swing window titled "Queue Simulation". The interface has a purple background. At the top, there are input fields for "Number of Queues:" (value 2), "Number of Clients:" (value 4), and "Simulation Time:" (value 15). To the right, there is a "Current Time:" label with a value of 6, and a dropdown menu set to "SHORTEST_QUEUE". Below these are two range input fields: "Arrival Time: between 2 and 7" and "Service Duration: between 2 and 4". A "Start Simulation" button is centered below the range inputs. At the bottom, there are five vertical queue containers labeled "Queue 1", "Queue 2", "Queue 3", "Queue 4", and "Queue 5", followed by a "Waiting Clients" area. Queue 1 contains two client entries: (0 5 2) and (3 6 4). Queue 2 contains one entry: (1 6 3). The other queues and the waiting area are currently empty.

8. Conclusions

Reflecting on the queue management system project, I've identified several areas for improvement and opportunities for further skill enhancement. This project has been invaluable in reinforcing my knowledge and skills in software development, highlighting both my strengths and areas for growth.

- **Thread Usage:** Implementing concurrent processing with threads has been essential for managing multiple tasks and servers simultaneously. By utilizing threads, the system can handle task dispatching, server management, and user interface updates concurrently, improving responsiveness and performance.
- **Thread-safe structures:** To mitigate potential concurrency issues such as race conditions and data inconsistency, I've incorporated thread-safe data structures like `AtomicInteger` in critical sections of the code. Atomic integers ensure that operations such as incrementing and decrementing shared variables are performed atomically, preventing race conditions and maintaining data integrity.

- Algorithm Design: Throughout this project, I've gained experience in designing algorithms to manage the queue, optimize server utilization, and handle task dispatching efficiently. These algorithms involve balancing the workload among servers, prioritizing tasks based on arrival times and service times, and minimizing wait times for clients.
- Data Structures: I've leveraged various data structures such as Lists, Queues, and AtomicInteger to manage tasks, servers, and queue states effectively. Understanding the strengths and weaknesses of different data structures has been crucial in optimizing the performance of the queue management system.
- User Interface Development: Employing design patterns like the Model-View-Controller (MVC) architecture has been instrumental in organizing the codebase and separating concerns effectively. This architectural pattern has facilitated modularity, maintainability, and scalability in the project.

By leveraging threads and thread-safe structures effectively, the queue management system can handle concurrent operations efficiently while ensuring data consistency and thread safety. This approach enhances the system's robustness and scalability, enabling it to effectively manage queues and servers in real-time scenarios.

Future Developments:

- Performance Optimization: Enhance the efficiency of task dispatching and server management algorithms to handle larger workloads and improve overall system performance.
- Visualization Enhancements: Implement visualizations to display real-time queue evolution, server utilization metrics, and task processing statistics, providing users with more comprehensive insights into the system's operation.
- Advanced Queueing Strategies: Introduce advanced queueing strategies such as priority-based task scheduling or dynamic server allocation algorithms to optimize resource utilization and minimize client waiting times.
- Integration with External Systems: Enable integration with external systems or APIs to fetch real-time data on incoming tasks or server performance metrics, allowing for more informed decision-making and adaptive system behavior.
- Automated Alerts and Notifications: Introduce automated alerting mechanisms to notify administrators or operators of critical events or system anomalies, such as server failures, queue overflows, or prolonged client wait times, enabling proactive intervention and problem resolution.
- User Authentication and Access Control: Implement user authentication mechanisms and access control features to ensure secure access to the queue management system, with role-based permissions for administrators, operators, and clients.

9. Bibliography

- 1) *Teach Yourself Java in 21 days* – Laura Lemay :
<https://www.cs.cmu.edu/afs/cs.cmu.edu/user/gchen/www/download/java/LearnJava.pdf>.
- 2) *What are Java threads?* - https://www.w3schools.com/java/java_threads.asp
- 3) *Threads in 10 minutes* -
https://www.youtube.com/watch?v=r_MbozD32eo&ab_channel=CodingwithJohn
- 4) *AtomicInteger in Java* - <https://www.digitalocean.com/community/tutorials/atomicinteger-java>
- 5) *Java Swing tutorial*:
https://www.youtube.com/watch?v=9XJicRt_FaI&t=1s&ab_channel=BroCode
- 6) *Lectures*