UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

University of Westminster

# Object Oriented Programming

5COSC019C

# Documentation Report

Name: Wijesinghe Arachchige Nihinsa Lineli Wijesinghe

UOW Number:20515180

IIT Number:20230039

Module leader: Sir Guhanathan Poravi

Group No: 22

# Table of Contents

# Table of figures

## 1.2 README File

The README File is included in the GitHub repos(linked in 1.7)

Google Drive Link: README File

### Real-Time Event Ticketing System with Advanced Producer-Consumer Implementation

#### Introduction

The Ticketing System Simulation is a multi-component program created to effectively manage ticket sales through concurrent and object-oriented approach. It has built from an Angular frontend for user interaction, a Spring Boot backend for managing business logic, and a Java Command Line Interface (CLI) for system configuration. The system guarantees thread safety, appropriate capacity management, and real-time updates while enabling vendors to distribute tickets into a pool that customers can retrieve tickets from.

### Setup Instructions

#### Prerequisites

- **Java**: Version 17 or above
- **Node.js**: Version 16 or above
- **Angular CLI**: Version 19.0.4
- **Maven**: For building the Spring Boot backend
- **Git**: For cloning the repository

### Building and Running the Application

1. **Clone the Repository**:

   ```
   git clone <repository-url>
   cd ticketing-system-simulation
   ```

2. **Build and Run the Backend**:

- Navigate to the backend directory:

  ```
  cd backend
  ```

- Build the Spring Boot application:

  ```
  mvn clean install
  ```

- Run the application:

  ```
  java -jar target/ticketing-system-backend.jar
  ```

3. **Build and Run the Frontend**:

   - Navigate to the frontend directory:

     ```
     cd ../frontend
     ```

   - Install dependencies:

     ```
     npm install
     ```

   - Run the Angular application:

     ```
     ng serve
     ```

4. **Run the CLI**:

   - Navigate to the cli directory:

     ```
     cd ../cli
     ```

   - Compile and run the CLI:

     ```
     javac CLI.java
     java CLI
     ```

## Usage Instructions

### Configuring and Starting the System

1. **Open the UI**: Navigate to http://localhost:4200 in your web browser.
2. Fill in the form:
    - **Total tickets per vendor**: The number of tickets each vendor can release.
    - **Maximum ticket capacity of the ticket pool**: The total number of tickets allowed in the pool(Ticket-pool size).
    - **Ticket release rate**: The rate at which tickets are released into the pool.
    - **Customer retrieval rate**: The rate at which customers retrieve tickets from the ticket pool.
    - **Number of vendors**: The total number of vendors releasing tickets.
    - **Number of customers**: The total number of customers retrieving tickets.
3. **Start the system**: Press the **Start** button to initiate the simulation.

### UI Controls

- **Configuration form:** Allows users to configure the system first
- **Start Button**: Begins the simulation with the provided parameters.
- **Stop button**: Terminates the simulation.
- **Ticket availability**: Shows the current number of tickets in the pool.
- **System Logs**: Displays real-time logs of the system's operations.

# 1.3 Diagrams

## 1.3.1 Class diagram

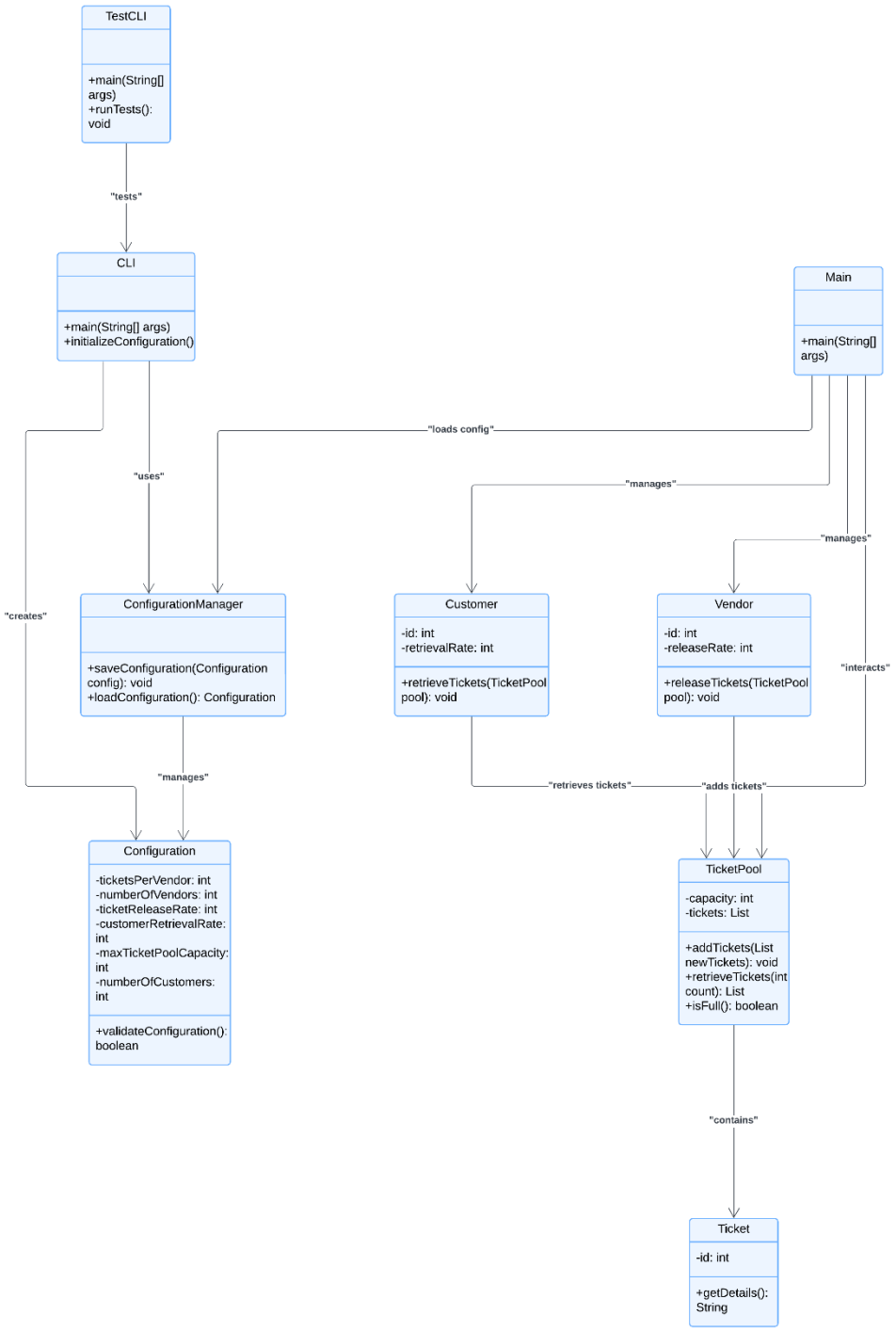


*Figure 1- Class Diagram*

### 1.3.2 Sequence diagram

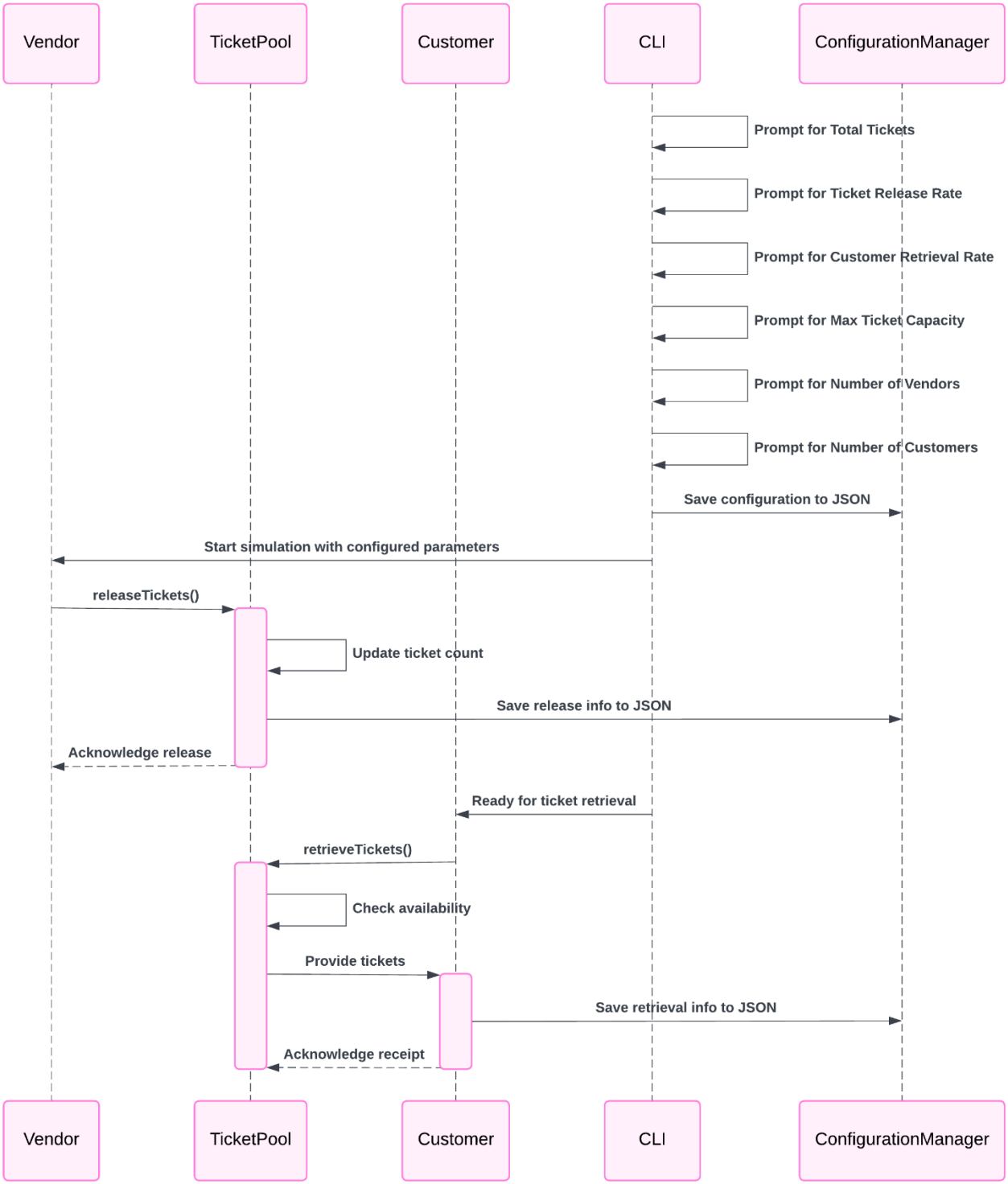


*Figure 2 - Sequence Diagram*

## 1.4 Testing report

| Test case - ID | Scenario | Input parameters | Expected result | Actual result | Issues found |
|---|---|---|---|---|---|
| 01 | Single vendor releases tickets and a single customer buys a ticket | Tickets per vendor: 10, Vendors: 1, Ticket release rate: 2, Customer retrieval rate: 1, Max ticket pool: 20, Customers: 1 | Customer successfully retrieves available tickets without exceeding their limit | passed | None |
| 02 | Multiple vendors issuing and multiple customers purchasing tickets simultaneously | Tickets per vendor: 10, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: 5, Max ticket pool: 50, Customers: 10 | Tickets distributed correctly among customers without exceeding ticket pool limits | passed | None |
| 03 | Ticket pool reaches maximum capacity | Tickets per vendor: 50, Vendors: 2, Ticket release rate: 10, Customer retrieval rate: 5, Max ticket pool: 100, Customers: 5 | System prevents vendors from adding more tickets when pool reaches maximum capacity | passed | None |
| 04 | Ticket pool size smaller than tickets per vendor * number of vendors | Tickets per vendor: 50, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: | System throws validation error message: “MaxTicketCapacity must be>(the total | passed | None |

| | | | | | |
|---|---|---|---|---|---|
| | | 2, Max ticket pool: 90, Customers: 5 | number of tickets per vendor*number of vendors) . Please re-enter a valid value" during CLI configuration | | |
| 05 | Retrieval rate exceeds release rate | Tickets per vendor: 20, Vendors: 2, Ticket release rate: 2, Customer retrieval rate: 5, Max ticket pool: 100, Customers: 5 | Tickets are retrieved without errors, and the pool decreases correctly | passed | None |
| 06 | Concurrent ticket retrieval and release | Tickets per vendor: 30, Vendors: 3, Ticket release rate: 10, Customer retrieval rate: 10, Max ticket pool: 300, Customers: 10 | Thread safety maintained, no race conditions or data inconsistencies observed | passed | None |
| 07 | Invalid parameters in CLI configuration (e.g., negative values, strings instead of integers) | Tickets per vendor: -10, Vendors: 2, Ticket release rate: t, Customer retrieval rate: 5, Max ticket pool: 100, Customers: 5 | System throws appropriate error message: "Invalid input. Please enter an integer:" and prompts for correction | passed | None |
| 08 | Invalid parameters in CLI configuration (integer 0) | Tickets per vendor: -10, Vendors: 2, | System throws appropriate error message: "Please enter a | passed | None |

| | | | | | |
|---|---|---|---|---|---|
| | | Ticket release rate: 0, Customer retrieval rate: 5, Max ticket pool: 100, Customers: 5 | positive integer larger than 0:" and prompts for correction | | |
| 09 | Maximum ticket capacity exceeded by a single vendor | Tickets per vendor: 60, Vendors: 1, Ticket release rate: 10, Customer retrieval rate: 2, Max ticket pool: 50, Customers: 5 | Vendor is blocked from adding tickets beyond the pool's capacity | passed | None |
| 10 | Angular frontend retrieves real-time updates from Spring Boot backend | Tickets per vendor: 20, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: 2, Max ticket pool: 100, Customers: 5 | Frontend displays accurate ticket availability and system status | passed | None |
| 11 | Stress test with maximum number of customers and vendors | Tickets per vendor: 50, Vendors: 10, Ticket release rate: 20, Customer retrieval rate: 50 Max ticket pool: 1000, Customers: 100 | System operates smoothly under heavy load without crashing | passed | None |
| 12 | Unauthorized access to ticket pool operations via backend | Tickets per vendor: 20, Vendors: 2, | Unauthorized requests are denied and logged | passed | None |

| | | | | | |
|---|---|---|---|---|---|
| | | Ticket release rate: 5, Customer retrieval rate: 2, Max ticket pool: 100, Customers: 5 | | | |
| 13 | Angular UI invalid input handling (e.g., invalid customer ID) | Tickets per vendor: 20, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: 2, Max ticket pool: 100, Customers: 5 | UI displays error messages and prevents invalid operations | passed | None |
| 14 | API response times | Tickets per vendor: 20, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: 2, Max ticket pool: 100, Customers: 5 | API responses under 500ms for normal operations | passed | None |
| 15 | Vendor-specific operations | Tickets per vendor: 20, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: 2, Max ticket pool: 100, Customers: 5 | Vendors can only operate within their allocated limits without affecting others | passed | None |
| 16 | Concurrency limits | Tickets per vendor: 30, Vendors: 3, Ticket release rate: 10, Customer retrieval rate: | System handles maximum thread utilization without deadlocks or starvation | passed | None |

| | | | | | |
|---|---|---|---|---|---|
| | | 10 Max ticket pool: 300, Customers: 10 | | | |
| 17 | Backend data inconsistency simulation | Tickets per vendor: 20, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: 2, Max ticket pool: 100, Customers: 5 | Backend recovers gracefully from crashes without data loss | passed | None |
| 18 | Frontend stress test | Tickets per vendor: 50, Vendors: 10, Ticket release rate: 20, Customer retrieval rate: 50 Max ticket pool: 1000, Customers: 100 | UI handles high API request volume without crashing | passed | None |
| 19 | Edge-case input tests | Tickets per vendor: 1, Vendors: 1, Ticket release rate: 0, Customer retrieval rate: 0, Max ticket pool: 1, Customers: 1 | System handles extremely large/small parameter values appropriately | passed | None |
| 20 | Session management | Tickets per vendor: 20, Vendors: 2, Ticket release rate: 5, Customer retrieval rate: 2, Max ticket pool: 100, Customers:2 | User sessions timeout as expected, and re-authentication is required | passed | None |

## 1.5 Demonstration video

Given below is the link to the demonstration video of this real-time event ticketing system. It shows the following:

- The configuration process.
- Implementation of all the test case scenarios
- Demonstration of the system running with multiple vendors and customers
- Concurrent ticket additions and purchases
- The connected UI

Google Drive Link : Demo Video

## 1.6 Assumptions

- Every vendor adds, and every customer purchases the same kind of ticket from the same type of event. This eliminates the need for extra layers of complexity and maintains the focus on threading and synchronization.
- In addition to the 04 parameters mentioned in the coursework specification, the CLI prompts the user for the number of vendors and customers during the initial configuration.
- Total Tickets per vendor:

Defines how many tickets in total a single vendor may release to the TicketPool.
This number is distributed across vendors evenly.
Unless specifically constrained, vendors can release tickets at the given rate without running out because their supply is infinite beyond this fixed figure.

- Ticket Release Rate:

controls how quickly tickets are added to the TicketPool.
Over the course of runtime, the release rate remains consistent.

- Customer Retrieval Rate:

controls the speed at which users can retrieve tickets from the pool, indicating the system's demand side.

Customers do not retrieve tickets in bulk; they do so individually.

- Max Ticket Capacity:

serves as a buffer to keep incoming tickets from flooding the TicketPool. If this limit is reached, vendors can not to release tickets into the pool.

The maximum ticket capacity should always >= Total Tickets per Vendor * Number of Vendors

If this capacity is surpassed, the system queues tickets.

- Number of Vendors:

decides how many separate threads or entities will be releasing tickets into the pool, which scales the system.

Every vendor works independently and releases tickets at the same pace and they do not interfere or compete with one another's ticket release activities.

- Number of Customers:

determines the number of independent threads or entities that will be using pool tickets to scale the system.

Unless application logic specifies otherwise, customers are presumed to have limitless demand.

## 1.7 GitHub Repo

Two separate Repos were used for the front end and the back end of this project:

Front End- Frontend-Angular.git

Back End- Real-Time-Event-Ticketing-System-.git

## 1.8 LinkedIn Learning Certifications

LinkedIn Profile: [lineli-w]

**Advanced Java: Threads and Concurrency:**


**Angular Essential Training:**

**Spring Boot 2.0 Essential Training:**


**Spring Framework in-depth:**

**Creating API Documentation:**